Thesis

Machine Learning Techniques for Charged Particle Tracking at the ATLAS Experiment

AMROUCHE, Cherifa Sabrina

Abstract

The Large Hadron Collider (LHC) uses proton-proton collisions to probe the fundamental building blocks of matter. Each collision produces thousands of particles scattering away from the detector center at nearly the speed of light. Reconstructing the trajectories of particles is a crucial task in most physics analysis. However, due to the rise in the number of simultaneous proton-proton interactions at the High Luminosity LHC (HL-LHC), the current tracking techniques will be the dominant component in CPU requirements. This thesis proposes the extension of existing as well as the design of novel Machine Learning (ML) approaches for the tracking of particles in the ATLAS experiment. We propose to describe and extend the similarity search problem in particle tracking through Approximate Nearest Neighbors (ANNs). In this context, the distance between data points is redefined with a tracking aware metric learning model termed TrackNet. Additionally, ANNs and metric learning models are evaluated on the TrackML dataset and on the ATLAS Inner Tracker Phase II dataset. We propose the Dynamic Tracking Linkage (DTL) clustering [...]

Reference

AMROUCHE, Cherifa Sabrina. *Machine Learning Techniques for Charged Particle Tracking at the ATLAS Experiment*. Thèse de doctorat : Univ. Genève, 2021, no. Sc. 5553

DOI : 10.13097/archive-ouverte/unige:152041
URN : urn:nbn:ch:unige-1520412

UNIVERSITÉ DE GENÈVE FACULTÉ DES SCIENCES

Département de physique nucléaire et corpusculaire Professeur Tobias Golling

Machine Learning Techniques for Charged Particle Tracking at the ATLAS Experiment

THÈSE

présentée à la Faculté des sciences de l'Université de Genève pour obtenir le grade de Docteur ès sciences, mention interdisciplinaire

par

Sabrina Amrouche

d'Algérie

Thèse n° 5553

GENÈVE

Atelier d'impression ReproMail

2021

UNIVERSITÉ
DE GENÈVE

FACULTÉ DES SCIENCES

DOCTORAT ÈS SCIENCES, MENTION INTERDISCIPLINAIRE

Thèse de Madame Cherifa Sabrina AMROUCHE

intitulée :

«Machine Learning Techniques for Charged Particle Tracking at the ATLAS Experiment»

La Faculté des sciences, sur le préavis de Monsieur T. GOLLING, professeur associé et directeur de thèse (Département de physique nucléaire et corpusculaire), Monsieur A. SALZBURGER, docteur (Organisation européenne pour la recherche nucléaire, CERN, Suisse), Madame T. MONTARULI, professeure ordinaire (Département de physique nucléaire et corpusculaire), autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 7 avril 2021

Thèse - 5553 -

Le Doyen

N.B. - La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "Informations relatives aux thèses de doctorat à l'Université de Genève".

À ma mère, à mon père

Abstract

The Large Hadron Collider (LHC) uses proton-proton collisions to probe the fundamental building blocks of matter. Each collision produces thousands of particles scattering away from the detector center at nearly the speed of light. Reconstructing the trajectories of particles is a crucial task in most physics analysis. However, due to the rise in the number of simultaneous proton-proton interactions at the High Luminosity LHC (HL-LHC), the current tracking techniques will be the dominant component in CPU requirements.

This thesis proposes the extension of existing as well as the design of novel Machine Learning (ML) approaches for the tracking of particles in the ATLAS experiment. We propose to describe and extend the similarity search problem in particle tracking through Approximate Nearest Neighbors (ANNs). In this context, the distance between data points is redefined with a tracking aware metric learning model termed TrackNet. Additionally, ANNs and metric learning models are evaluated on the TrackML dataset and on the ATLAS Inner Tracker Phase II dataset. We propose the Dynamic Tracking Linkage (DTL) clustering algorithm to process the output of the TrackNet model and to retrieve the final particle trajectories. This tracking inspired algorithm encapsulates physics constraints in its pairwise distance as well as a trained classifier that acts as an automatic stopping criteria.

Restricting the standard ATLAS seeding to ANNs buckets allows to reach the peak performance of 88% with a CPU time of 5ms per bucket. Moreover, the similarity search tracking environment enables the finding of additional tracks as compared to the standard ATLAS tracking specifically in low P_T regions. The tracking time per bucket is further reduced when replacing combinatorial tracking by the TrackNet+DTL chain. Finally, a voting cluster shape convolutional (VCS-Conv) model is proposed as the first attempt to include raw images of activated pixels in an ML based tracking model. This heterogeneous approach (raw images and data points) is shown to outperform classical models based on raw information only.

Résumé

Au grand collisionneur de hadrons (LHC), l'exploration des constituants fondamentaux de la matière se fait au moyen de collisions de proton-proton. Des milliers de particules émanent de chaque collision et se propagent dans le détecteur à presque la vitesse de la lumière. La reconstruction de la trajectoire de ces particules est une des tâches les plus importantes dans l'analyse physique. Cette tâche se voit cependant se compliquer et dominer les besoins en CPU dans le cas du nouveau High-luminosity LHC où le nombre de collisions simultanées est nettement augmenté.

Cette thèse propose de nouvelles techniques ainsi que l'extension d'algorithmes existant de reconstruction des trajectoires sur la base de l'apprentissage automatique (ML). La recherche des similarités dans de grandes bases de données est étendue à la problématique de la reconstruction des trajectoires de particules à travers les techniques de recherche approximatives du plus proche voisin. Pour cela, la similarité entre voisins est redéfinie selon des contraintes physiques par un modèle d'apprentissage de métrique qu'on appellera TrackNet. La combinaison des deux approches est évaluée sur deux ensembles de données : TrackML et l'ATLAS Inner Tracker Phase II. Pour le regroupement final des trajectoires, nous proposons un nouvel algorithme de clustering appelé Dynamic Tracking Linkage (DTL). Cet algorithme comprend des contraintes physiques dans la distance qu'il utilise ainsi qu'un classifieur pour l'arrêt automatique de l'algorithme.

Par ailleurs, la formation des premières pistes (seed) de particule dans des groupes approximatifs de plus proche voisin permet d'atteindre une performance de 88% avec un temps CPU de 5ms par groupe de point avec notamment la reconstruction de trajectoires supplémentaires dans les régions de faible PT. De plus, nous proposons un modèle de convolutions (VCS-Conv) en mesure de traiter les images des pixels activés lors du passage des particules. La combinaison de données numériques et d'images permet de dépasser les performances de modèles basés seulement sur l'une d'entre elles.

Contents

Introduction

Can machine learning help in finding the trajectories of charged particles produced in high energy experiments? This question is at the origin of the research presented in this document. Today, based on this research and many more studies, we know that indeed, machine learning *can* help in reconstructing charged particle tracks. Helping in the context of the present research refers to improving the time spent in finding the tracks. Unquestionably, the current tracking techniques reach peak performances on the present data volumes. However, the combinatorial approach relied upon becomes unusable with the future increase of the data intake. In turn, the data volumes are produced by increasing the probability of particles colliding in the LHC. This increase has then a significant impact on our discovery potential, i.e. our understanding of the universe. This abstract chain is illustrated in Figure 1.

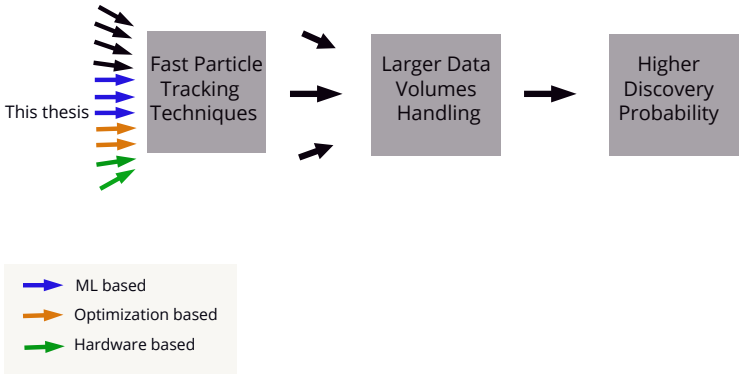


Figure 1: Illustration of the *big picture* on particle tracking and this work. Dark arrows represent additional techniques not falling in any of the mentioned categories.

Fast tracking techniques are needed in order to process the large data volumes. Optimization techniques rely on using fixed or dynamic thresholds to reduce the search space. These thresholds are generally found through the manual evaluation of many thousands of parameters. The current tracking techniques rely in large proportions on optimization. Machine learning based techniques rely on the implicit patterns found by mining large datasets. The resulting models produce automatic decisions with very little manual optimization. Hardware based techniques implement tracking algorithms on dedicated hardware such as GPUs and FPGAs. The algorithms can be machine learning models or optimization/combinatorial techniques. Additionally, a tracking technique can combine algorithms from the three different categories. In this work, we investigate primarily machine learning based techniques with some extensions towards hardware implementations.

Concepts such as discovery potential, particle collision, tracks are intro-

duced in Chapter 1. The experimental apparatus where the previous concepts take place is detailed in Chapter 2. Relevant definitions as well as standard algorithms for particle tracking are summarized in Chapter 3. An introduction to machine learning (ML) and the models used or referred to in this document is presented in Chapter 4. Fast search techniques lay at the intersection of optimization and computational theory and represent an important component of this work. They are detailed in Chapter 5. The TrackML challenge which represents the first significant attempt at using machine learning to solve particle tracking is discussed in Chapter 6. The author contributed to the set up of the challenge as part of the ATLAS author qualification project.

Finally, the core of this work, i.e. the implementation, test and discussion of various ML concepts on two different particle collision datasets are demonstrated in Chapters 7, 8, 9 and 10.

1

Experimental Particle Physics

Particle physics is the branch of physics that studies the fundamental constituents of matter and their interactions. A sub-field of particle physics is referred to as High Energy Physics (HEP) because many of the fundamental constituents do not exist in nature and *high* energies are required to produce them. In fact, particle physics dynamics are governed by quantum mechanics. And it is quantum mechanics that is responsible for the most accurate physics theory known as the Standard Model. In this chapter we briefly introduce the Standard Model and the fundamental notions necessary for the understanding of this work. A complete account of the standard model can be found in [1][2].

1.1 The Standard Model of particle physics

The framework that describes the elementary laws of nature is known as the Standard Model (SM). More specifically, it encapsulates theories on the properties of fundamental particles and their various interactions. Over the last decades, these theories have been tested and verified through extremely sophisticated experiments such as the ATLAS experiment detailed in Chapter 2. Undeniably the SM does not explain every physical phenomena and still has limitations such as an explanation for the dark-matter content in the universe or a quantum theory of gravity. Despite these limitations that constitute the focus of current research in physics, the SM, with the discovery of the higgs boson in 2012, is the most successful and robust physical model.

According to the standard model, the universe is a product of **particles** and **forces**. The interplay of these two, creates everything. Figure 1.1 shows the elementary particles as described by the Standard model. An elementary particle can be seen as the smallest dot in the universe, i.e. it cannot be fragmented.

The first classification of particles (also starting from the left in Figure 1.1) is into quarks and leptons. Leptons, such as electrons, are particles that interact through the electromagnetic and weak forces while quarks interact through the strong force. There are **four** fundamental forces :

- The gravitational force. A long-range force, negligible compared to other forces. It is not described by the Standard Model.
- The electromagnetic force. Also long-range, this force acts on electrically charged particles.
- The weak force. At the level of subatomic particles, it is for example responsible for radioactive decay.

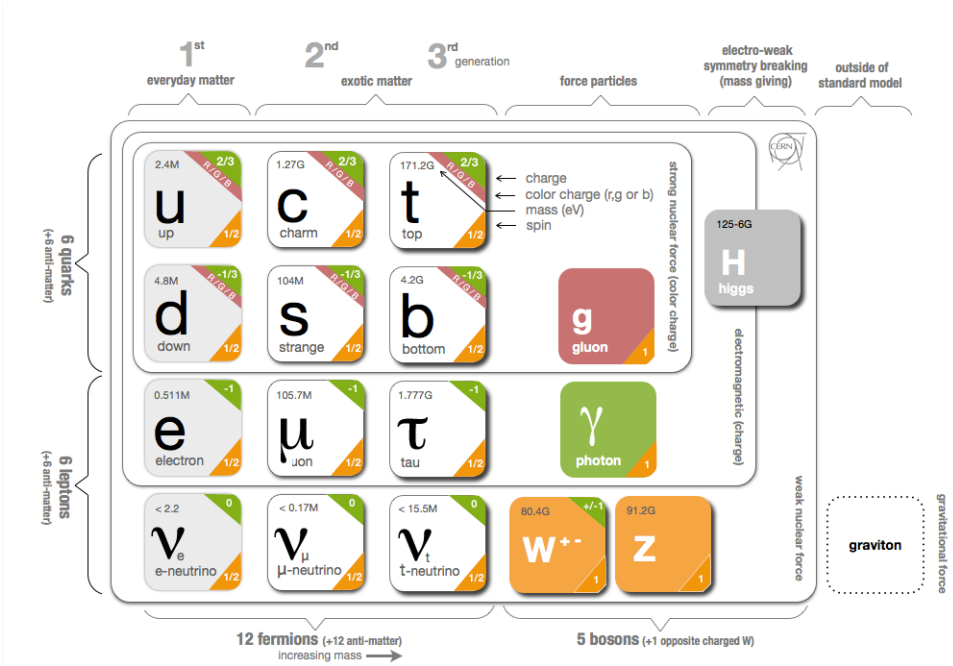


Figure 1.1: Particles of the Standard Model [3]

- The strong force (as its name suggests, the strongest force) holds the nucleus together, binding quarks into protons or neutrons. Because of the strength of this force, quarks cannot exist freely in nature (unlike leptons) but rather form hadrons[1]. Quark and gluons are also refered to as partons.

Three[2] of the fundamental forces result from the motion of *force carrier* particles called bosons. Each fundamental force has its own force carrier: gluons carry the strong force, photons carry the electromagnetic force and W and Z bosons carry the weak force. One of the basics of the Standard Model framework is that the elecromagnetic force and the weak force are manifestations of the same underlying force : the elecroweak force. This unification allowed to describe all the particles and the force carriers accurately except for their mass which was not accounted for in the model's equations. To address this shortcoming, theorists Brout, Englert and Peter Higgs proposed a mechanism by which an invisible field gives a mass to particles that interact with it. This field is now called the Higgs field and the particle associated with it, a Higgs boson. One of the major goals intended for the Large Hadron Collider (LHC) was the observation of this particle and therefore, the validation of the Standard Model as a successful framework. A goal fully achieved when the ATLAS and CMS experiments jointly announced the observation of the Higgs boson in 2012.

Colliding particles in a controlled environment is the technique used to study their fundamental properties. This is how the atom nuclei was first discovered and a century later, the Higgs boson observed.

[1] Hadronization is a process where quarks and gluons become hadrons.
[2] There are theories that propose a *graviton* force carrier for gravity.

1.2 Colliding Particles

Physicist Charles-Augustin de Coulomb, in 1787 stated that "The magnitude of the electrostatic force F between two point charges q1 and q2 is directly proportional to the product of the magnitudes of charges and **inversely proportional to the square of the distance between them**." This means that if the distance between two protons is halved, their repulsion force is quadrupled. When protons are accelerated in the LHC, their beam is constantly focused such that the distance separating protons is at its smallest when reaching the collision point (approximately 10^{-5}cm). When particles collide, the repelling force is at its maximum causing particles to *scatter* away. Figure 1.2 shows two accelerated particles approaching a collision point.

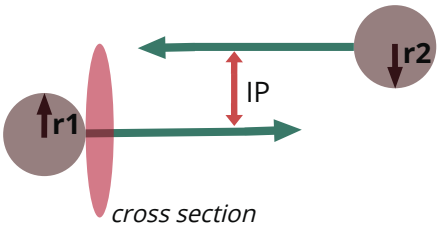


Figure 1.2: Colliding particles : Impact parameter and collision cross section.

The impact parameter (IP) depicted in the figure is the hypothetical closest distance between the two particles centers. The area shown by the red circle in Figure 1.2 is called collision *cross section* [4]. If the center of an approaching particle (in opposite direction) is within this area, then a collision takes place. Depending on the value of the IP, the trajectory of the particles after the collision is deflected or scattered due to an energy (momentum) transfer. Studying this alteration of particles trajectories is a powerful tool to understand the different forces and structures at play. If the scattering of the particle is kinematic only (a change in direction only), it is called an *elastic* scattering. If, however, the particle emits a new particle[3] or absorbs energy, the scattering is called *inelastic*. The quark was discovered by means of deep inelastic scattering (DIS) where the structure of hadrons is probed using leptons as illustrated in Figure 1.3.

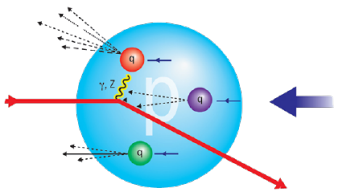


Figure 1.3: Deep Inelastic proton-electron scattering process. An electron (red arrow) is scattered away from a proton generating boson γ, Z that in turn interacts with the quark.

During a collision, the interaction producing the largest energy is referred to as *hard-scatter* [5]. The hard-scatter occurs when quarks and gluons collide, transferring an excessive amount of energy and mass. Therefore, the study of these hard-scatter probes is crucial to the understanding and exploration of the

[3]The energy is converted into mass via $E = mc^2$

properties of matter (especially that partons cannot be studied directly). The *observables* generated from a hard-scatter can be radiation, high transverse[4] momentum particles and jets. Often, gluons and quarks create showers of particles through hadronization and fragmentation. These collimated charged and neutral particles form a *jet* as shown in Figure 1.4. The radiation patterns of jets encode the properties of the quarks and gluons that initiated the hadronization. The study of the content and properties of jets represent an important focus of high energy physics experiments. A comprehensive introduction on jets can be found in [6][7].

When two bunches of protons are accelerated, the hard-scatter collision point is referred to as primary vertex. It is the most energetic collision point. Less energetic (and more common) collision points are referred to as pileup vertices. When the rate of collisions increase, multiple proton-proton collisions occur simultaneously. In the context of the High Luminosity LHC for example, the number of these pileup events reaches 200. The pileup collisions, being less energetic, are less interesting to study. An important complication however is the *contamination* of the total energy by these *uninteresting* collisions. The mitigation of pileup interactions is therefore necessary for a correct event reconstruction.

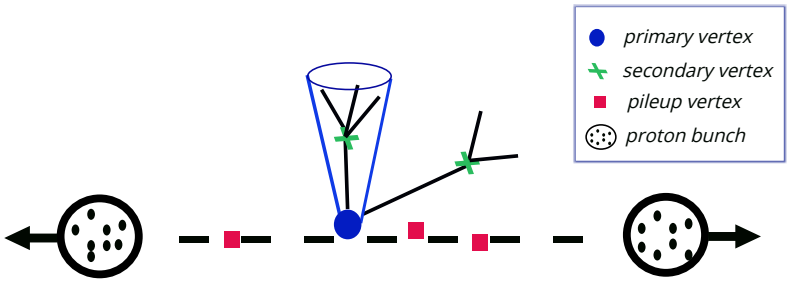


Figure 1.4: Collision and vertices. The cone shaped decay is a jet.

Figure 1.4 also shows the decay of particles originating from the primary vertex into new particles called *secondary* particles. If the decay takes place at a relatively important distance from the primary vertex, the originating tracks are referred to as *displaced* tracks. Displaced tracks are generally harder to reconstruct as they are not aligned with the interaction point. Secondary particles hold important information about the primary particles and their detection (and reconstruction) is the first technique used to analyze a collision.

1.3 Particle detection

The first detection technique used to explore high energy collisions is a passive one. The detector layers that are used to measure particles minimally affect their trajectories through electromagnetic interactions. This *tracking* technique is therefore able to detect charged particles only. Multiple detection layers are placed along the path of the particle. At every layer[5], the particle interacts with the material loosing a small fraction of its energy in the form of an electrical charge. Using the position of the deposited charge, a tracking algorithm is able to accurately recover the original track of the particle.

[4] The momentum component that is transverse to the beam.

[5] Detection modules can can stop functioning resulting in a missed particle measurement.

Although detection layers are designed to interact as little as possible with the passing particle, *multiple scattering* occurs along the trajectory resulting in a deviation from the ideal path. The energy transfer of this interaction is negligible due to the small mass of the passing particle but the successive small angle deviations add a randomness component to the trajectory. This constitutes an additional challenge to account for in the design of a tracking algorithm.

Figure 1.5 illustrates the detection mechanism for the reconstruction of trajectories. Detection layers are placed such that many measurements of the particle trace are recorded. Associating the different measurements together to recover a particle trajectory is the first step towards the analyses of a collision. Multiple scattering effects are added in the first and last detection layers of Figure 1.5 (b), slightly deviating the measurements from the actual track.

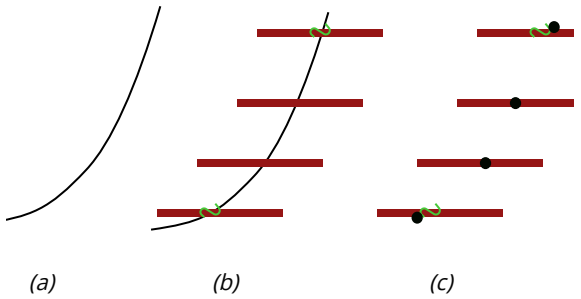


Figure 1.5: (a) Particle trajectory. (b) Detector layers for the measurement of charged particles. Multiple scattering is illustrated on the first and last layers (green curve). (c) Result of the detection or how the particle is finally *seen* through the detector.

The charge deposited by a particle can *activate* more than one sensor. Figure 1.6 illustrates a charged particle traversing a layer of sensors and producing a charge in 3 of them. The charge cluster that is created has a shape of (2,2) in the illustrated (u,v) coordinates system. From the shape of the charge cluster and the detection layer/sensors properties, it is possible to extract the incidence angle (or inner angles) of the particle along the two illustrated axis. The incidence angles, depending on the sensor resolution, provide an additional information for the tracking application.

The particle physics notions introduced in this chapter are independent from the experiment or detector layout. In the next chapter and throughout this work, we will specifically focus on the ATLAS detector and the data it records.

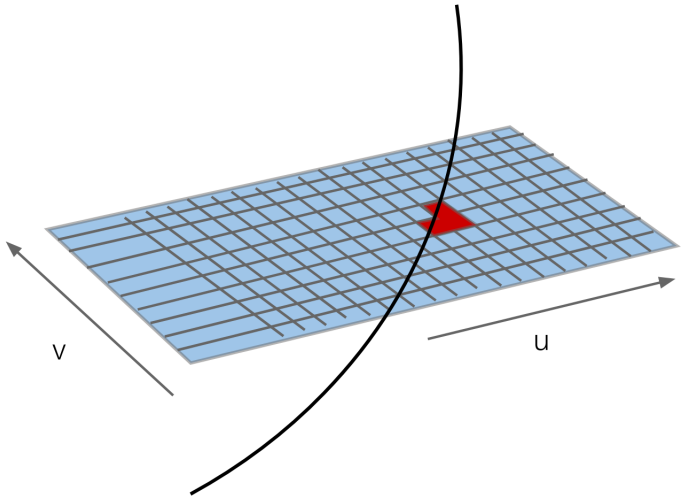


Figure 1.6: Charge deposited by a particle traversing pixel sensors.

Bibliography

[1] Peskin, Michael. An introduction to quantum field theory. CRC press, 2018.

[2] Pais, Abraham. "Inward bound: of matter and forces in the physical world." (1986).

[3] Andrew Purcell, Go on a particle quest at the first CERN webfest,2012 `https://cds.cern.ch/record/1473657`.

[4] Scattering Experiments and Classical Theory of Atom–Atom Scattering , in Theory of Molecular Collisions, 2015, pp. 1-18 DOI: 10.1039/9781782620198-00001

[5] Butterworth, Jonathan M., Guenther Dissertori, and Gavin P. Salam. "Hard Processes in Proton-Proton Collisions at the Large Hadron Collider." Annual Review of Nuclear and Particle Science 62 (2012): 387-405.

[6] KLAUS. RABBERTZ. JET PHYSICS AT THE LHC: The Strong Force Beyond the Tev Scale. SPRINGER, 2018.

[7] Particle Data Group Collaboration,Review of Particle Physics, Chin. Phys.C40(2016)100001

2

The ATLAS experiment at the Large Hadron Collider

At the entrance of the CERN complex stands an exhibition center that explains the Large Hadron Collider (LHC) mission to the public. While visitors stand there, an electronic voice states that the LHC mission is to discover what we, as humans, are made of and how our universe started to exist. And indeed, the LHC, with every particle collision is allowing us to reach back in time to as few as 10^{-14} seconds after the big bang. This chapter presents details of the LHC acceleration complex, the particle collisions with their underlying physics and the different experiments.

2.1 The Large Hadron Collider

The LHC [1] accelerates protons in opposite directions and brings them to collision in locations known as *interaction regions*. The protons are grouped in 2808 bunches of $N_c = 1.15 \cdot 10^{11}$ proton each. These bunches of protons are kept on their circular paths with the magnetic field created by approximately 2000 superconducting dipole magnets. The collision rate at the LHC is determined by the number of bunches and their proton count. The higher these two quantities, the more collisions take place. The main limiting factor of the number of bunches and the number of protons per bunch is the cooling of the magnets. In order to reach a superconducting state, the magnets are brought to extremely low temperatures: 1.9 K (-271.3C), colder than the 2.7 K (-270.5C) of outer space). Since accelerating charged particles emits more power due to synchrotron radiation, the cooling of the magnets (among other factors) imposes a maximum of $N_c = 10^{11}$ protons per bunch.
Increasing the rate of collisions is necessary since many interesting physics processes have a tiny probability to occur. The order of the cross section σ being at the picobarn ($10^{-28}m^2$), the LHC has to operate at a high collision rate per unit of area and time. This quantity is called *instantaneous luminosity* $\mathcal{L}$:

$$\mathcal{L} = f \frac{N_b \; n1 \; n2}{A} \tag{2.1}$$

N_b being the number of bunches per beam, $n1$ and $n2$ the number of protons in the colliding bunches, f the revolution frequency and A the overlapping area of the bunches. The design instantaneous luminosity of the LHC is equal to $10^{34} cm^{-2} s^{-1}$ [2].

The integrated luminosity, $\int \mathcal{L} dt$ is used to quantify the expected number

of occurrences N of a physics process after a certain measurement time:

$$N = \int \mathcal{L}\, dt \cdot \sigma \tag{2.2}$$

Thus, the increase in luminosity is proportional to the increase of process occurrences, i.e. number of events which in turn translates to a larger dataset. The LHC schedule is composed of running periods (Runs) and long shutdowns (LS) in between to enable its upgrade. Figure 2.1 summarizes the LHC and the future High Luminosity LHC (HL-LHC) runs. The LS2 is expected to end in 2021 as Run3 starts.

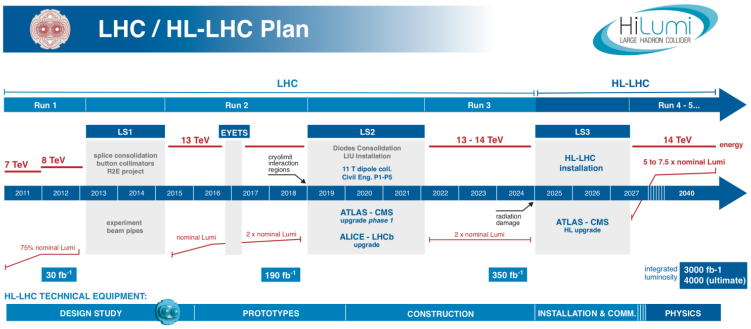


Figure 2.1: LHC and HL-LHC timeline [3]

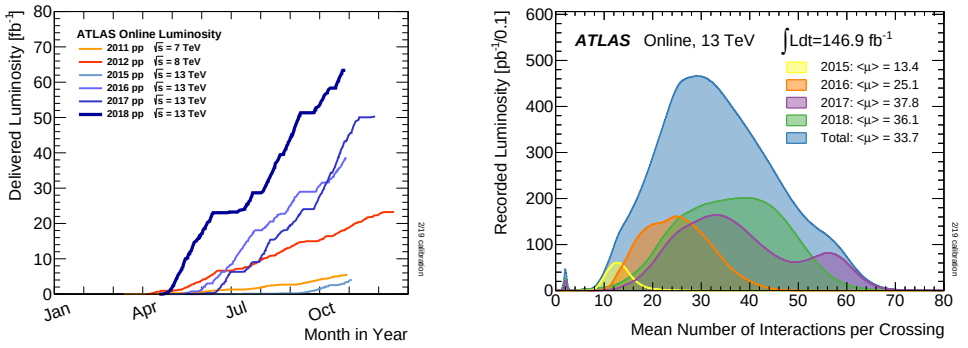
The different upgrades primarily target an increase in luminosity as seen in Figure 2.2(a) which is a key factor for a particle accelerator. The HL-LHC, expected to start in 2025, will reach a high luminosity by increasing $n1$ and $n2$ and decreasing A in equation 2.1 by squeezing the beams. The higher the luminosity, the higher the probability of collision of protons pairs from opposite bunches. The number of simultaneous proton collisions in the same bunch crossing is referred to as *pileup*. This distribution is often characterized by its mean μ. Figure 2.2(b) illustrates the mean number of interactions per crossing recorded in stable beam collisions of the LHC Run 2.

In the HL-LHC and in the context of this work, the pileup studied has a mean number of pileup collision of $\mu = 200$. There are two categories of pileup. The in-time pileup that occurs simultaneously with the physics process of interest (signal event or hard scatter) and the out-of-time pileup that represent residual proton collisions from previous events. Both pileup collisions are regarded as contamination of the signal and have to be mitigated through the detector tracker.

The proton collisions are recorded and analyzed in four major experiments along the LHC as illustrated in Figure 2.3.

Each experiment has built a detector targeted at one or multiple physics aspect. The design of a detector in number or structure of the layers through which particles are captured is tuned towards a particle type. Different detector designs yield different precision measurements that in turn affect the final physics results. The design of a detector although seemingly motivated purely by its physics reach has multiple constraints some of which are budgetary, safety related or technology imposed. Both the software used to collect and analyze data and the hardware are under constant investigation and upgrade.

ATLAS [5] and CMS [6] are the two largest detectors at CERN. In 2012, the two associated experiments announced jointly the discovery of the Higgs Boson. Their mission now mainly focuses on probing new physics through the LHC record luminosity. ALICE [7] covers the studies of quark-gluon plasma created in heavy ion collisions. LHCb [8] focuses in precision measurements



(a) (b)

Figure 2.2: Evolution of (a) $\int L\ dt$ over the years and (b) of the interaction per crossing evolution [9]

of CP violation and rare decays of bottom and charm hadrons. To reduce the impact of cosmic rays on the LHC and its detectors the whole complex has been installed 100 metes below the Swiss-France frontier.

In this document, we will focus on the ATLAS experiment although the data analysis principles are detector agnostic.

2.2 The ATLAS Detector

The ATLAS detector [5] measures properties of particles produced in proton-proton collisions to probe the nature of particle interactions and search for new phenomena. Figure 2.4 illustrates the layered detector structure. The protons are brought to collide in the center of the detector. The innermost system known as inner detector (ID) is used to retrieve charged particle trajectories with their precise parameters such as momentum and vertex position. The particles, when leaving the ID will interact with the calorimeter material producing showers. These showers are induced to measure the particles energy through the Electromagnetic or the Hadronic interactions.

Not all particles are detected that way. Some neutral particles such as neutrinos will not leave traces in the tracker nor interact with either calorimeters. Muons for example have a small interaction probability with the calorimeters and for their detection, ATLAS has a special standalone tracker known as the Muon Spectrometer (MS). Figure 2.5 illustrates the behavior of different particle types through the detector component. At the tracker level, it is not possible to distinguish a muon from a charged pion. The MS easily identifies muons since they are the only particles to reach it. At the end of the track reconstruction, both are processed as continuous tracks to the calorimeter. Inside the tracker and before any reconstruction, the particles are in fact a collection of points or hits in the detector rather than curves.

Contrary to what might appear, not all particle collisions are registered and analyzed. The data acquisition is regulated by a strict entry door known as the trigger. It is an online[1] mechanism that selects which collision is recorded and stored. In the next sections, these different detector systems are explained.

2.2.1 Detector Coordinate System

To describe the many aspects of particles emanating from high energy collision, a coordinate system definition is required. This system provides a standard

[1] At the time of the collision or as close to it as physically feasible

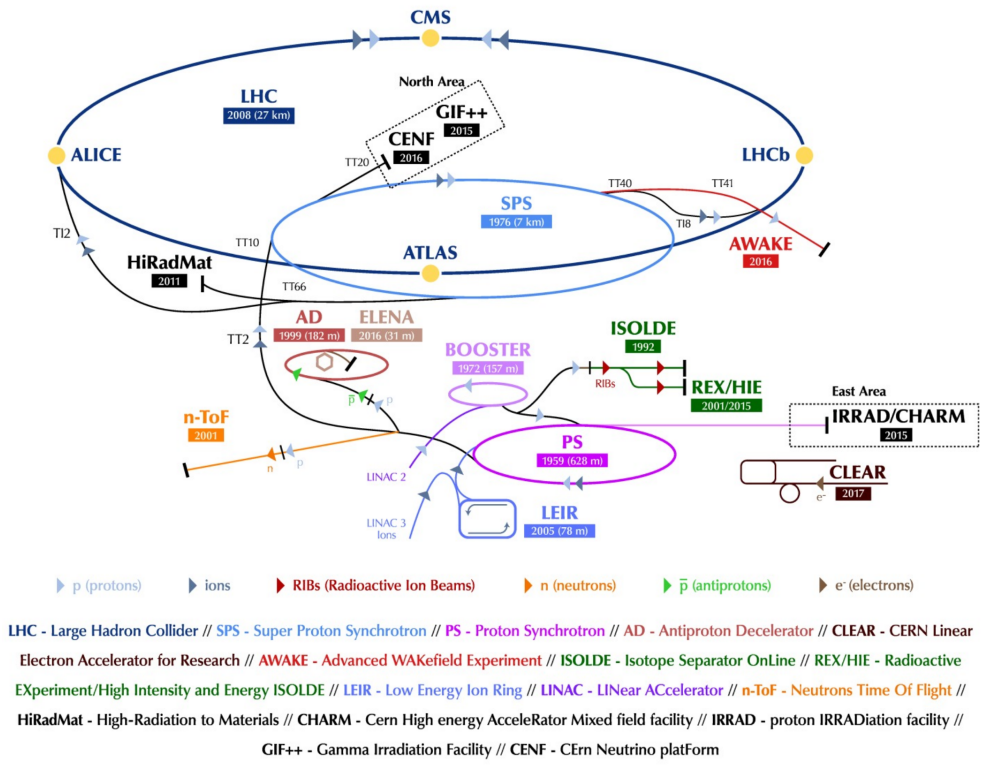


Figure 2.3: CERN accelerator complex [4]

across the experiment and enables efficient description of its complex structure. Some of the quantities defined in this section will be used in different chapters to reference specific particle behavior.

The origin of the coordinates system is chosen to be the nominal interaction point. The z axis is chosen to be aligned with the beam axis and the polar θ angle is defined with respect to it. The positive x axis points to the center of the LHC ring and the positive y axis points upwards. The xy plane is transverse to the beam axis. Quantities such as the transverse momentum P_T of a particle are defined with respect to this plane :

$$P_T = \sqrt{p_x^2 + p_y^2} = |p| \ \sin\theta \tag{2.3}$$

P_T is proportional to the inverse curvature of a particle trajectory in the xy plane. Low momentum particles curve more in the transverse plane and are more challenging to find independently of the technique used (standard tracking or machine learning based).

The second important quantity is the rapidity of a particle y:

$$y = \frac{1}{2} \ \ln\left(\frac{E + p_z}{E - p_z}\right) \tag{2.4}$$

where E is the energy of the particle and p_z its momentum along the z axis. When the particles are considered massless (given its momentum), the quantity in Eq. 2.4 is approximated by the geometric quantity *pseudo-rapidity*:

$$\eta = - \ \ln\left(\tan\frac{\theta}{2}\right) \tag{2.5}$$

The pseudo-rapidity η is commonly used to refer to detector regions. Finally, the detector is split into Barrel Region (BR) sections, cylinders around the z axis, and End-cap Regions (ER) sections, wheels perpendicular to the z axis.

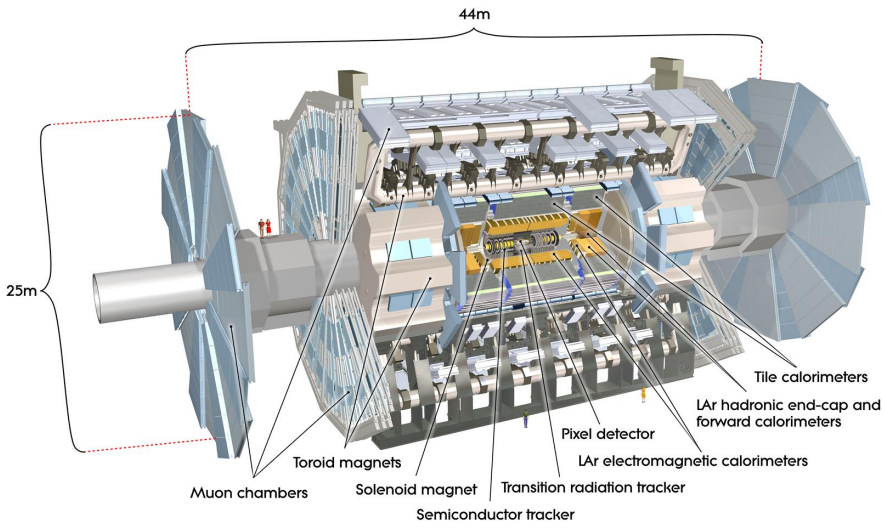


Figure 2.4: ATLAS detector schema [10]

2.2.2 The Inner Detector

The Inner Detector uses three different technologies to provide the first accurate measurements of the particles. Figure 2.6 illustrates the different components and their dimensions. The ID is formed by three sub-detectors:

- Pixel Detector, immediately on top of the beam pipe, four pixel layers (Run 2) provide high precision 3D measurements of the hits.
- Semiconductor Tracker (SCT) surrounding the pixel layers also providing precise 2D measurements.
- Transition Radiation Tracker (TRT) is the outer most component of the track and uses straw tubes filled with a gas mixture. It allows the collection of a higher number of measurements, improving the momentum resolution at the expense of resolution and limited rate capability.

The insertable b-layer (IBL) was added to the Pixel detector for Run 2 and it is an example of the possible upgrades performed to enhance precision.

Pixel Detector

The pixel detector contains about 2500 modules and 92 million channels. The IBL is composed of planar and 3D silicon pixel sensor technology and the earlier three layers are made of silicon planar sensors. A traversing charged particle ionises the material of the pixel module and create pairs of electrons/holes along the trace. The opposite charges then drift apart under the applied voltage and can be registered as a hit. This hit position are derived from the deposited charge. The process of determining the hit positions from the charge is called clusterisation. Since the timestamp[2] cannot resolve particles from one bunch crossing in time, it is possible that the charge in the modules belong to more than one particle. This is referred to as hit merging and disentangling the corresponding clusters (especially in dense environments) is done with a neural network. With its four layers, the pixel detector provides at least 4 track hits for the reconstruction. These innermost hits dominate the impact parameter and the vertex resolution.

[2] An additional time measurement on the activated modules would allow to disentangle measurements created at different times from different particles.

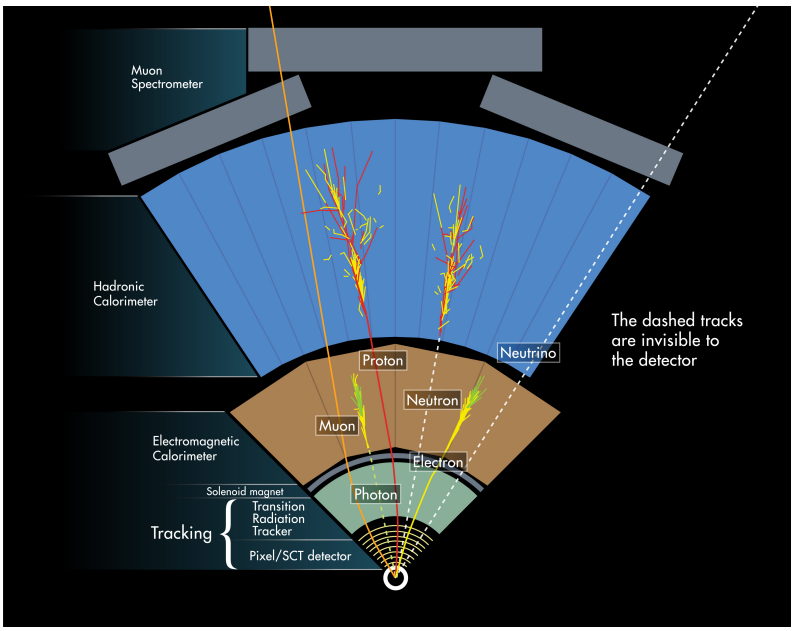


Figure 2.5: Detection stages of different particles [11]

Semiconductor Tracker

The Semiconductor Tracker (SCT) is composed of four cylindrical layers in the barrel region and two end-caps each containing nine disk layers. SCT has a binary readout that limits the spatial resolution along the sensitive direction to 23 µm. The initial resolution was increased with a sandwich like structure where each SCT module has two sensors rotated by $+/-40$mrad with respect to each other. This double sided structure provides a 1D measurement from each sensitive plane. They are then combined along the correct direction (of the plane) to the global coordinate space. SCT comprises 988 modules in each of the two end-caps and 2112 in the barrel with a total of 6.2 million readout channels.

Although disappearing in the future upgrade of the detector (discussed in the next Section), the final (and largest) segment of the current detector is the Transition Radiation Tracker.

Transition Radiation Tracker

The Transition Radiation Tracker (TRT) are essentially 4mm diameter tubes filled with gas called drift tubes. When a particle passes through a tube, it interacts with the gas mixture and an electron shower is created towards the center of the tube. The shower arrival time distribution is used to estimate the *drift time* measurement effectively resulting in a small circle (of drift radius) which the particle crossed at given location. The exact location retrieval is performed by track reconstruction.

In the barrel, the straws are 144cm long parallel to the beam axis. In the end-caps they have a length of 37cm. The TRT contributes to the reconstruction in two major aspects:

- The stacking of the tubes allows more than 30 hits per track and thus contributes massively to the resolution of the momentum.
- The space between the drift tubes is filled with a transition radiation material that emits X-rays when traversed by a particle. Electrons emit larger amounts of photons compared to other particles and so a collection

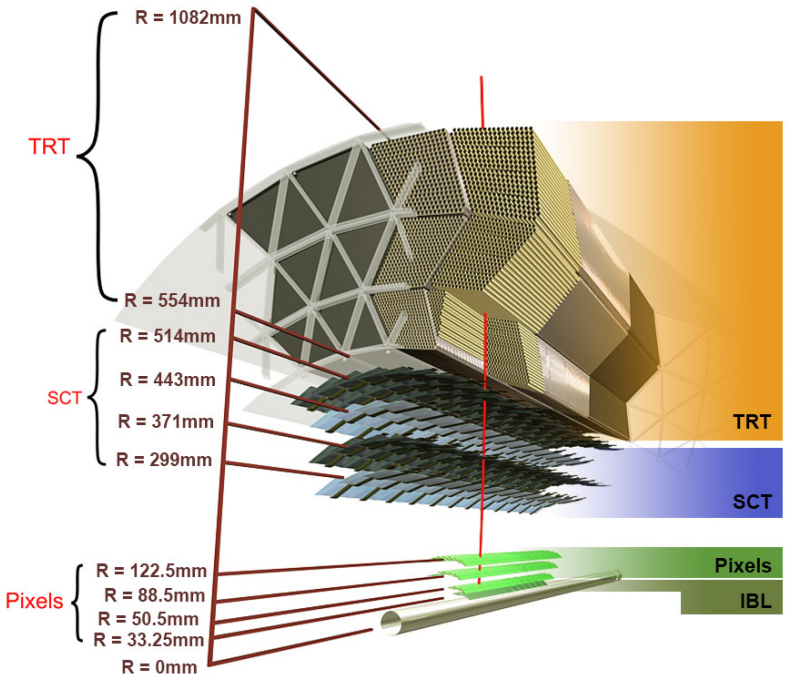


Figure 2.6: A schematic view of the ATLAS Inner Detector (ID) [12]

of these X-rays allows a complementary particle type identification to the calorimeters.

Another example of detector upgrade is the change of gas mixture composition of the drift tubes during LS1. Ar replaced the more expensive Xe.

Before moving on to the outer most components of the ATLAS detector, namely the Calorimeters and the Muon Spectrometer, it is important to highlight the upgrade of the ID for Phase II and HL-LHC.

2.2.3 Detector Upgrades

The Phase-II upgrade is planned to be installed by 2024 (Figure 2.1) and with it an increase in instantaneous luminosity that the current ID cannot handle. The different detector and physics aspects that will be affected by this upgrade are:

- The increase in radiation damages the sensors and causes current leakage[3] causing the hit efficiency[4] to drop,i.e. a sensor is no longer able to detect the passage of a particle.
- The granularity of the current SCT and TRT sub-detectors cannot cope with the increase in pileup. A high occupancy can significantly degrade performance. For example, resolving merging particles will not be feasible (degraded tracking performances).

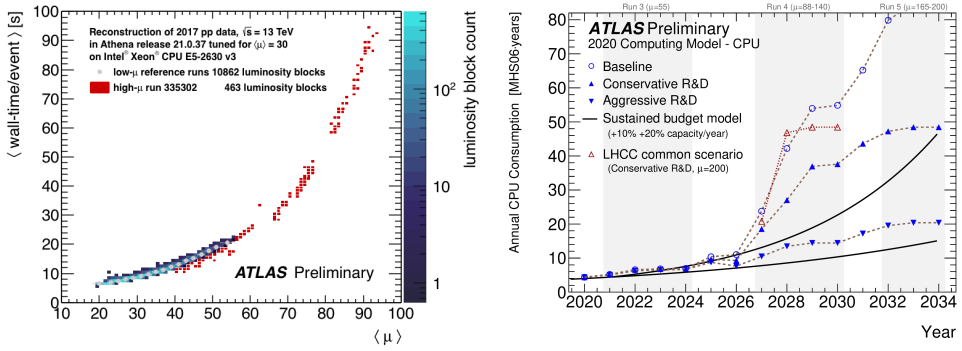
Figure 2.7(a) shows the implications of running on higher pileup (introduced in Section 1.2) while keeping the same detector granularity and tracking performances. The expected pileup for HL-LHC is μ=200. The exponential scaling of the wall-time[5] per event shows the inability of current algorithms

[3] A leakage current is an unwanted electric current, it increases as a function of the total radiation dose.

[4] The detector hit efficiency has to be 100%

[5] Also known as elapsed real time. It is the actual time that a program takes to run.

to handle the the expected volume of data. In parallel, Figure 2.7(b) shows various scenarios for future computing models. The blue distributions assume significant improvement and development work in the simulation software, in reconstruction and in event generation.



(a) (b)

Figure 2.7: (a) The effect of an increase in the average number of interactions per bunch crossing ($< \mu >$) on the reconstruction wall time per event. (b) Estimated CPU resources needed for the years 2020 to 2034 for both data and simulation processing.

The two plots shown in 2.7 sparked significant R&D to tackle the reconstruction performances. The reduction of CPU consumption is also the main focus in this PhD thesis. Yet, even if the reconstruction (and simulation) performances are substantially improved, it is first necessary for the detector to upgraded in order to cope with the higher luminosity.

ATLAS Phase-II Inner Tracker

The ATLAS Phase II Inner Tracker (ITk) [14] is the replacement of the current ID. It is optimized to cope with future run conditions, specifically HL-LHC, without degrading the tracking performances.

The ITk, contrary to the ID, comprises two subsystems: a Strip Detector and a Pixel Detector (Figure 2.8). The latter extends the coverage to $|\eta| < 4$ with its five layers. The close by rings in the pixel detector are positioned to ensure a larger number of hits per layer thus improving tracking conditions. Additionally, the layers closest to the interaction point are *replaceable* as an anticipation to the extreme radiation environment expected at the HL-LHC.

The work presented throughout this document will solely revolve around the Inner Detector system where the task is to reconstruct charged particle tracks. The ITk layout presented in Figure 2.8 is used in Chapter 8 to demonstrate the proposed approach.

2.2.4 Electromagnetic and Hadronic Calorimeters

The calorimeter task is to measure precisely the energy of the particles and thus correctly identifying the amount of missing energy which is crucial for searching for new physics. As shown in Figure 2.5, the Calorimeters are capable of detecting more particles than the ID (charged and neutral). It does so by absorbing the energy of particles when interacting with the material through a showering process. The shower type and interaction holds the signature of the particle type. The ATLAS calorimeters use sampling technology where an absorbing layer is followed by an active (measuring) layer and the initial particle energy is obtained through the sum of all energy deposits.

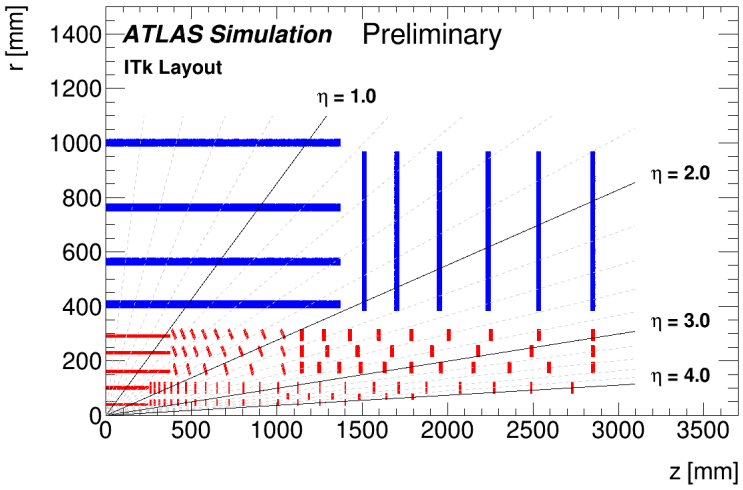


Figure 2.8: The ITk layout with pixel layers in red and strip layers in blue.

A liquid-argon (LAr) sampling Electromagnetic Calorimeter (ECAL) is used to measure light electro weak particles such as electrons, positrons and photons.

The Hadronic Calorimeter (HCAL) uses steel absorbers and scintillator-tile sampling technology as well as copper/LAr sampling technology to capture hadronic particles such as protons, neutrons and pions.

2.2.5 The Muon Spectrometer

The Muon Spectrometer (MS) is the last piece of the ATLAS detector and its role is the reconstruction of muons that barely interact with the Calorimeters (loosing small fraction of their energy). Fast Muon reconstruction is an important requirement of triggering (detailed in the next section 2.2.6). For this purpose, the MS has two subsystems for precision tracking: Monitored Drift Tubes (MDT) and Cathode Strip Chambers (CSC) and two additional detectors for fast momentum estimation: Resistive Plate Chambers (RPC) and Thin Cap Chambers (TGC). TGC have a higher granularity for copping with the higher multiplicity in the forward region. The fast measurement allows also the distinction between muons from different bunch crossings.

2.2.6 The Trigger

The LHC generates in a single hour the volume of data[6] that Facebook accumulated and stored since its creation. That is, data on billions of user activities (images and videos included). Contrary to social media, interesting[7] hard scatter events constitute only a small fraction of the detector readout. Concretely, only one bunch crossing in 10^5 produces a potentially interesting event. The selection of this interesting event is the job of the trigger system. An event must pass a two level decision process to be stored for offline processing.

- Level-1trigger (L1) [13] a hardware based trigger that reduces the data rate from 40 MHz to 100 kHz. The decision making occurs within 2.5

  The L1 is deployed on custom-built electronics and uses coarse inputs from the Calorimeters (L1Calo) and the MS (L1Muon) to make approximate esti-

[6] One petabyte of collision data per second.

[7] In standard reconstruction, uninteresting events are low P_T proton scattering

mations on objects of interest in selected events. The L1 does not make use of any information from the ID due to the readout time. An example of selection criteria is the presence of leptons which are rarely present in uninteresting events. The presence of a lepton is a flag for retaining an event and makes up 20% of the L1 bandwidth. The L1 detects the regions where the interesting object has been detected known as regions of interest (RoI). Once an event is accepted, it is passed to the HLT along with the coordinates of the RoI. The HLT is a 28000 CPU core farm that runs fast software on the L1 RoI to extract the event physics signature. The algorithms used are similar to the ones run in offline analysis, only the HLT focuses on the refined RoI of the L1 which decreases significantly the response time. The output of the HLT is processed by the Tier-0.

Tiers are levels for computer centers [15]. Tier-0 is the first set of computer clusters that process and store the raw data directly from the LHC. It is the CERN data center, physically located inside the CERN complex. Next in the grid hierarchy is the Tier-1 with its 13 computer centers distributed across three continents. They are responsible for the storage on disk and tape of the raw and reconstructed LHC data. Tier-2 are 155 universities or scientific institutions providing computing units for analysis as well as performing simulations. Tier-3 represents the end user level and can be a single laptop performing data analysis.

2.3 Monte Carlo Simulation

Similarly to a robot being trained in simulations before its first step on Mars, collisions are modeled in accurate Monte Carlo simulations before the LHC is turned on. The role of simulations in High Energy Physics experiments is to *predict* the physics produced during a collision. This prediction is based on theories of the established Standard Model of particles (see Chapter 1). Discrepancies between these theories and what is collected from a real collision are indicators of an unexpected phenomena : new physics. In fact, despite the complexity and sophistication of the LHC, the *data* that it produces is non interpretable without confronting it to simulation. Furthermore, a new particle or physics phenomena could be hidden in the data today but unless it is aligned with the corresponding simulation (thus theory), it cannot be discovered.

Monte Carlo methods repeatedly sample from the underlying probability distribution to approximate an expected value. These techniques are well suited to simulate high energy physics, inherently stochastic, interactions.

The simulation of the collision and the particles created from it (their type, energy and so on) is carried in *generators*. They are the translation of theory equations and models into high energy physics events. The most popular generators are PYTHIA [15], HERWIG [17] and SHERPA [18]. In order to model a detector response to these generated events, an additional software is required. This software propagates the collision in a specific detector geometry (ATLAS for example) and models the passage of every particle through the detector layers. Simulations in ATLAS [20] use the Geant4 [19] software for exactly this purpose. An essential input to Geant4 is the detector description file that contains a detailed representation of the physical detector (even pipes and wires). The more details are put into the detector description, the more accurate and slow the simulation becomes. Simulations using Geant4 with a detailed physics description and the complex physics geometry, referred to as full simulation, have such an important run-time that they cannot be used in

many physics studies that require high statistics. Over the years, many *fast* simulation variants have been developed to accelerate the slowest components in the full simulation. For example, the slowest part in the full simulation (75%) is due to the simulation of electromagnetic particles [20]. The fast G4 simulation software accelerates this process by a factor of three when replacing the simulation of these particles showers with *pre-simulated* ones. The first application of machine learning techniques for showers simulation [21] have proposed the use of generative models as a faster alternative to parameterized simulations.

ATLFAST-II was developed in order to speed-up the full simulation process even more (factor 100). It comprises the fast ATLAS Tracking Simulation (Fatras) for the simulation of the inner detector and muon system and the Fast calorimeter Simulation (FastCaloSim) for the calorimeter. Fatras keeps an accurate description for sensitive detector parts only and simplifies the rest of the detector. The extrapolation engine [22] used in the offline track reconstruction algorithm performs the propagation of the particles in this simplified geometry. Fatras provides the collection of simulated particles to the FastCaloSim package. FastCaloSim in turn, speeds up the simulation of particle showers in the calorimeter through parametrizations (histograms) of the longitudinal and lateral energy profiles estimated from 30 millions full simulation events [20].

As mentioned earlier, Fatras and FastCaloSim are designed and optimized for the ATLAS detector geometry. An interesting research direction is the experiment-independent fast simulation offered by ACTS (A Common Tracking Software) [16]. ACTS is a toolkit that performs track reconstruction using modern software paradigms while inherently enabling parallel architectures executions. It can be seen as a research and development platform for particle tracking. ACTS was used as the fast simulation engine for the TrackML Challenge (Chapter 6) and therefore the first dataset used in this work.

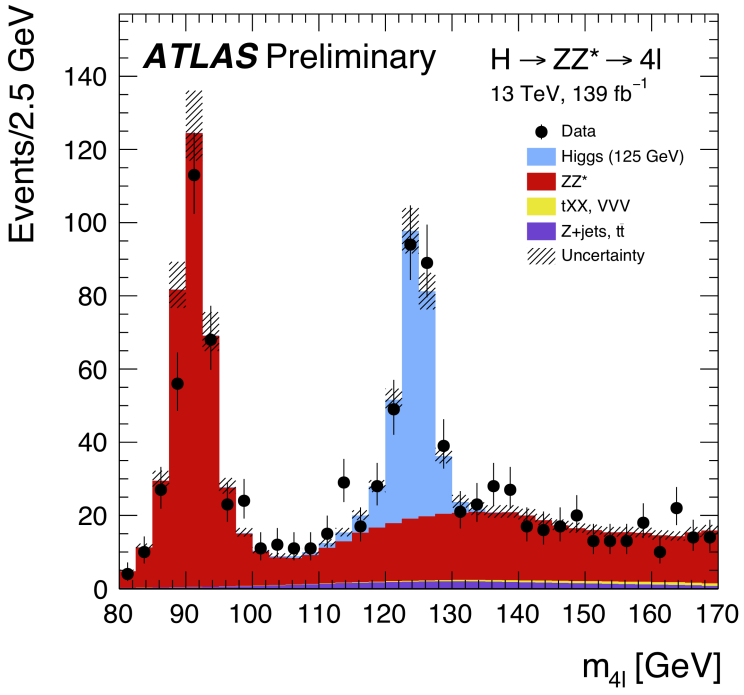


Figure 2.9: Distribution of the invariant mass of the four leptons selected using the full ATLAS Run-II dataset. The Higgs boson corresponds to the excess of events in blue.

The last piece of the simulation pipeline is called *digitization*. Once the particle interactions are modelled, the information on the energy deposits with the corresponding detector coordinates becomes available. Digitization converts this information into a format similar to the detector readout. Not only the charge measurements and coordinates are converted into voltages and currents (the actual detector output) but also low energetic physics is overlayed such as pileup (also pre-simulated for speed). After this last step and approximately 2 million lines of code, the output of the simulation reassembles in every aspect the data collected from the LHC and additionally contains all the information on the simulated physics. Figure 2.9 shows the agreement between collected data (black dots) and simulation where all the decays are labelled.

Bibliography

[1] L. Evans and P. Bryant,LHC Machine, JINST3(2008) S08001.

[2] LHC Design Report, Report number: CERN-2004-003-V-1, DOI: 10.5170/CERN-2004-003-V-1

[3] The LHC and HL-LHC timeline https://cds.cern.ch/record/1975962/

[4] E. Mobs. The CERN accelerator complex. Complexe des accélérateurs du CERN. https://cds.cern.ch/record/2684277

[5] ATLAS Collaboration,The ATLAS Experiment at the CERN Large Hadron Collider, JINST3(2008) S08003.

[6] CMS Collaboration,The CMS Experiment at the CERN LHC, JINST3(2008) S08004.

[7] ALICE Collaboration,The ALICE experiment at the CERN LHC, JINST3(2008) S08002.

[8] LHCb Collaboration,The LHCb Detector at the LHC, JINST3(2008) S08005.

[9] ATLAS Collaboration,Luminosity Results for Run2 https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResultsRun2

[10] Joao Pequenao, Computer generated image of the whole ATLAS detector, https://cds.cern.ch/record/1095924

[11] Joao Pequenao and Paul Schaffner, How ATLAS detects particles: diagram of particle paths in the detector. https://cds.cern.ch/record/1505342

[12] The ATLAS inner detector https://cds.cern.ch/record/2209070/plots

[13] The ATLAS Collaboration. Performance of the ATLAS Trigger System in2015. European Physical Journal C,77,2017

[14] The ATLAS Collaboration Collaboration,„Technical Design Report for the ATLAS ITk Pixel Detector,Tech. Rep. ATL-COM-ITK-2018-019, CERN, Geneva, 2018.https://cds.cern.ch/record/2310230

[15] The Grid: A system of tiers https://home.cern/science/computing/grid-system-tiers

[16] Sjöstrand, Torbjörn, et al. "An introduction to PYTHIA 8.2." Computer physics communications 191 (2015): 159-177.

[17] G. Marchesini, B.R. Webber, G. Abbiendi, I.G. Knowles,M.H. Seymour and L. Stanco, HERWIG: A Monte Carloevent generator for simulating hadron emission reactionswith interfering gluons. Version 5.1 - April 1991, Comput.Phys. Commun.67(1992) 465

[18] T. Gleisberg et al., SHERPA 1.alpha, a proof-of-conceptversion, JHEP02(2004) 056, [hep-ph/0311263]

[19] S. Agostinelli et al., Geant4 - a simulation toolkit, Nucl.Instr. Methods Phys. Res.A 506(2003) 250–303

[20] Aad, Georges, et al. "The ATLAS simulation infrastructure." The European Physical Journal C 70.3 (2010): 823-874.

[21] ATLAS collaboration. (2018). Deep generative models for fast shower simulation in ATLAS (No. ATL-SOFT-SLIDE-2018-1028). ATL-COM-SOFT-2018-182.

[22] A. Salzburger, The ATLAS Track Extrapolation Package,ATL-SOFT-PUB-2007-05 (2007)

[23] A Common Tracking Software https://acts.readthedocs.io/en/latest/

3

Track Reconstruction

Particle tracking is solving a puzzle. The challenge is recognizing which footprints are left by which object. Once all the pieces are associated with their doers, the map is complete and one can uncover all the paths and interactions.

In charged particle tracking, particles create a puzzle of traces (footprints). Due to technology constraints the traces left in the detector are atemporal. Particles passed through different locations at different moments in time but the output of an event has all the information of their passage at once making the traces ambiguous and highly dense[1]. Another challenge is the high luminosity environment which opens new physics horizons by increasing the rate of collisions and thus the number of traces. This chapter discusses the reconstruction of charged particle trajectories from the creation of traces in the detector to the final information extracted from each retrieved particle.

3.1 Tracking Notions

Before diving in into tracking algorithms some definitions are necessary:

Tracking efficiency

Along with the fake rate, the tracking efficiency or only "efficiency" is considered as the most important quantity. This is, because the understanding of the physics process requires to have an as complete-as-possible picture. The efficiency is the ratio between reconstructed tracks and actual tracks that pass predefined criteria. If a collision creates M particles and out of these N particles pass criteria such as having high enough momentum, then the efficiency is the fraction of N particles that are found by the algorithm.

$$Efficiency = \frac{N_{reco} \cap N}{N} \tag{3.1}$$

With N_{reco} the number of tracks found by the tracking algorithm. An ideal tracking algorithm would achieve perfect efficiency, i.e. reconstruct all particles in the sample N. A particle is considered found if the collection of its hits is good enough to estimate the track parameters.

A particle trajectory is defined with five parameters shown in figure 3.1. The ultimate goal of track reconstruction is to find the parameters that describe the particle closest to its origin with the smallest error. This is then further used to interpret the particle within the context of the physics event (vertex, particle type, etc). Particle parameters are :

[1] Detectors that allow for sufficient time resolution as well are only in development

- The transverse impact parameter d_0 is the distance of the point in the track closest to the interaction point (IP) in the x-y plane.
- The longitudinal impact parameter z_0 is the z coordinate of the point in the track closest to the IP.
- The polar and azimutal angles θ and ϕ, respectively.
- The ratio q/p of the reconstructed track charge q and momentum p.

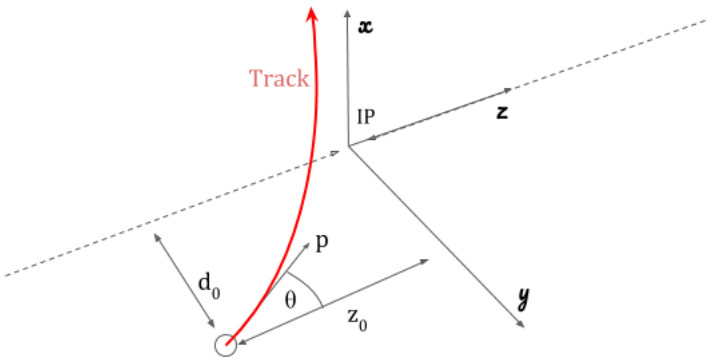


Figure 3.1: Visual insight into track parameters. IP represents the interaction point. The impact parameter representation is chosen assuming to be close to the vertex position, i.e. the actual origin of the particle.

The found tracks might not be the exact subset of real particles but close enough to retrieve similar properties. A matching criteria is introduced to define the minimum hit subset to consider a particle as found. In simulation data, the matching is performed either by looking at the truth content of the hits or by approximately estimating the truth physics parameters. It is worth noting that this is only possible in simulation where the truth information is available. There exist, indeed techniques to match tracks to found particles also in data taken in an experiment, where no truth information is available. These consist for example of tag probe methods, but are beyond the score of this work.

Fake rate

Because the efficiency is so important, a tracking algorithm might find more particles than actually exist only to minimise the probability of missing some, i.e. $|N_{reco}| \geq |N|$. A collection of hits that is not produced by a single particle but returned by the tracking algorithm is referred to as fake. Usually, if a track contains 50% and more of hits from different sources, then it is labelled as fake. The fake rate is therefore the fraction of fake hits combinations over the total reconstructed tracks.

$$Fake\ rate = \frac{N_{reco} - (N_{reco} \cap N)}{N_{reco}} \tag{3.2}$$

Fakes can be produced at different stages of the tracking algorithm but it is crucial that they are reduced before the final results. Fake tracks can cause an imbalance in the momentum/energy conservation hypothesis or point to wrong

vertices. Various procedures aim at identifying and removing fake tracks.

Holes

In tracking procedures, different track hypothesis are built and followed. When a measurement is expected by the algorithm to be on a certain sensor but there is none present, the measurement or rather its absence is referred to as hole. Possible causes for holes are inactive (faulty) sensors or intrinsic inefficiency although the detector condition information is integrated to avoid false hole flagging. Generally a good track should contain at most one hole.

Hit merging

In dense environment such as jets, the distance between particles is so small that the readout channels cannot distinguish between their hits. Particles share measurements or their hits get merged. This is different from the splitting of merged clusters as the distance between particles is much smaller in the case of hit merging. A hit can therefore be part of more than one track hypothesis (candidate).

3.2 From Detector to Space Points

Tracking charged particles in the inner detector is a fundamental task in a HEP experiment. The detectors are usually constructed such that, when a particle is created, it leaves traces on the modules of the detector and it is only through this traces, assembled together, that we can find back or reconstruct its full trajectory. A trace is the localisation of the particle in the detector. It is signaled by the readout channels that are activated by the passage of a charged particle (the particle deposits a charge[2]). A particle can activate multiple readouts that are grouped together into clusters with a connected component analysis. Those so called clusters are then used to build space points (hits) that represent the single location through which the particle is assumed to have traversed the detector layer. Figure 3.2 depicts the clusters produced by the passage of a particle through multiple layers. The procedure of deriving a single location to represent a cluster is called *clusterisation*. Depending on the detector characteristics, different clustering techniques can be deployed:

- Clustering using only digital readout: Pixels/Channels that are above a certain readout threshold are activated, but no further distinction between individual channels contributing to one cluster can be done. This is equivalent to a center of gravity algorithm in which the hit position is taken as the average of all pixels. If a large number of pixels are activated (long clusters), the algorithm crops those to retain only the main pixels.

- Clustering using analog information: if the detector is capable of determining (at least approximately) the amount of charge induced in a single readout channel, e.g. through time over threshold (ToT[3]) readout infor-

[2]The passing particle ionises the detector material and the charge is subsequently read out.

[3]The time the signal remains above a certain threshold, which is correlated with the amount of charge deposited.

Figure 3.2: Schematic view of multiple readout channels activated by a single particle passing through three pixel layers. Colors correspond to the amount of charge deposited in each pixel (yellow is highest). On the right, the output of the clusterisation is shown. From each cluster a single localisation is derived, indicated by the black cross. Each cross is called a hit or space point.

mation, this can be used to further refine the cluster position evaluation. In general, the charge (or ToT) is proportional to the path length of the particle within the readout cell, hence it can be used to interpolate the hit position.

A neural network to split merged clusters

To date, the only machine learning based algorithm in use in the ATLAS offline track reconstruction, is the identification of merged clusters. In dense environment, multiple particles can activate the same pixels. Figure 3.3 presents a drawing of such (frequent) scenarios. The highlighted pixels in the figure could be caused by one, two or more particles. The task of the neural network (NN) [1] is to split such clusters and assign each subgroup to the particle that created it. In practice, a set of ten neural networks are used. Each has a two hidden layer architecture and while the first model predicts the number of particles that caused the clusters, the remaining nine estimate the particles positions with errors. Prior to the use of an NN, the splitting of clusters could only be resolved iteratively at a later stage.

Once the clusters are split and space points formed, the next step is the building of seeds.

3.3 Building Seeds

This step reconstructs the first segment of a track, its seed. Any three points (triplets) that are aligned with a helix model can be investigated, becoming a seed. Concretely, if this triplet passes certain cuts, e.g. maximal transverse or longitudinal impact parameters or other simple geometrical requirements such as a minimum distance between hits, it is retained for the next stage. A seed is not necessarily built from the inner most hits. Seeds formed by strips hits are formed as well. Because the efficiency is such a prime concern, multiple seeds can be constructed for the same particle and therefore the same track road is followed multiple times. The optimal seeding algorithm, however, would

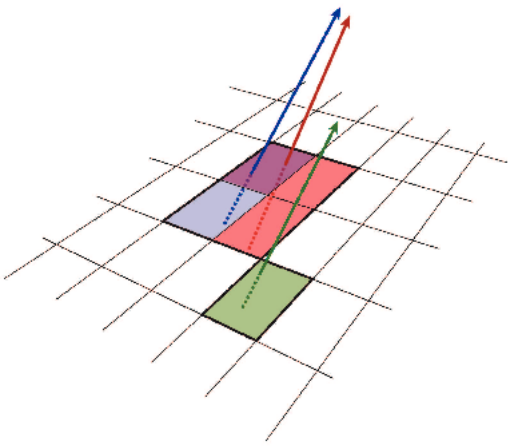


Figure 3.3: Illustration of multiple particles depositing charges in pixels. Particles are indicated by the arrows and the pixels activated by each arrow is shaded with its associated color [1].

produce exactly one significantly good seed per particle to be reconstructed. If a seed is not produced for a certain particle, it will not be retrieved in later stages (no back propagated information).

The choice of having triplets as starting points comes from the fact that three points are the minimal requirement to draw a helix (circle in xy and a line in z). Choosing a larger seed size could reduce the combinatorics needed. Chapter 9 presents a machine learning based alternative to building and filtering seeds early on.

Figure 3.4 shows the total number of seeds formed per bins in η and per event. As illustrated by the Figure, only a small fraction are used for the final tracks especially that a $\mu = 200$ event produces on average 4000 particles of interest.

Following the geometry of the detector (the number of hits produced), the seed multiplicity can be extremely high. Moreover, for every accepted seed an additional computational cost is added whether it is transformed into a track candidate (highest computational cost) or by checking against previous seeds.

After a triplet passed the first set of predefined cuts in the seeding, a fourth point is required[4] to confirm a seed quality before proceeding to the combinatorial filtering.

3.4 Combinatorial Track Finding

After a seed is formed and it has passed the seeding cuts, the Combinatorial Kalman Filter (CKF) will build a route starting from the seed and pointing to the next layers [3]. Hits compatible with this route are tested with the track hypothesis. The evaluation of the particle trajectory through the detector and magnetic field is a highly CPU intensive operation, hence it should ideally only be done for track candidates that are likely to describe trajectories for particles to be found.

A number of *cuts* are imposed on the seeds (routes) to maximise their purity such as a minimum momentum and a maximum displacement from the impact parameters. The cuts used in standard tracking are summarized in Table 3.1.

[4]This is specific to the ITk seeding strategy

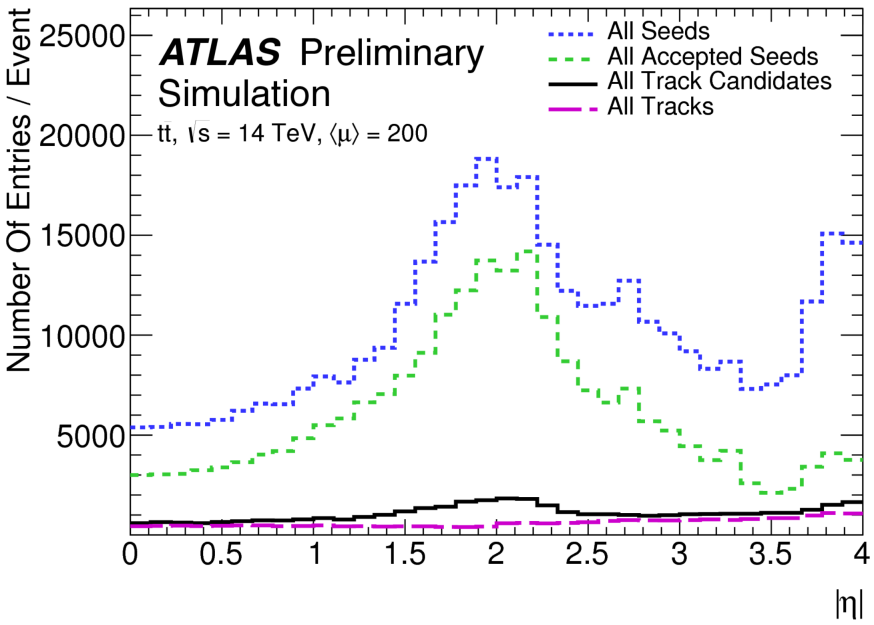


Figure 3.4: Overview of seeds distribution along the pseudo-rapidity η for the ITk layout (four points required to form a seed). Out of the total amount of seeds formed, the subset that passes basic criteria (d_0 or presence of holes for example) is accepted and within it, a much smaller fraction is used to form track candidates. Later, only few survive to make up the final track output. The surface between the magenta line and the blue one is spent in combinatorics not giving any particle.

This process is repeated with all hits found along the seed route, from layer to the next. Because of this combinatorial approach, a seed can give multiple routes and in the absence of an early filtering mechanism, the combinatorics (route possibilities) *explode* in an exponential way. Cuts are applied to discard tracks with not enough hits or too many holes. Similarly to the seeding, if no track candidate is built for a particle, it cannot be retrieved at a later stage.

The output is known as track candidates and as shown in Figure 3.4, they are close to the final output along most of the pseudorapidity range. As many combinations have been made, incorrect hit-track candidate assignments have been made. The ambiguity solving module addresses these cases.

Requirements	Pseudorapidity interval		
	$\eta < 2$	$2 < \eta < 2.6$	$2.6 < \eta < 4$
Pixel hits	≥ 1	≥ 1	≥ 1
Pixel+Strip hits	≥ 9	≥ 8	≥ 7
Holes	≤ 2	≤ 2	≤ 2
p_T $[MeV]$	> 900	> 400	> 400
$\lvert d_0 \rvert$ $[mm]$	≤ 2	≤ 2	≤ 10
$\lvert z_0 \rvert$ $[cm]$	≤ 20	≤ 20	≤ 20

Table 3.1: Minimal requirements for the default tracking depending on the pseudorapidity region for the planned ATLAS ITk detector [2]

3.5 Ambiguity Solving

In the ATLAS track reconstruction, an ambiguity solving routine runs over all track candidates that are sorted according to a quality score. The score derives from quantities such as the one presented in Table 3.1 where a track having a hole is more penalized than a complete track. A track fit, based on minimum least square estimation is run and measurements contributing disproportionately to the fit can be dropped from a track candidate or a poorly fitting track can be removed completely. Additionally, merging clusters are split with a neural network algorithm described in section 3.2. A final reconstruction fit is run using a very detailed detector material description to estimate the precise track parameters. This last step produces all the information needed from the tracker and is passed to the next sub-detectors. The scoring and cuts applied prior to this step are implemented to reduce drastically the number of candidates on which to run the final fit.

3.6 Faster Tracking

Figure 3.5 shows the evolution of the CPU time per event versus the average pileup. The dotted lines represent the Run-2 tracking with the standard cuts while the continuous lines represent the ITk layout. The principal differences naturally are due to the new detector layout that was designed for the increased pileup. Despite this, the total reconstruction time at $< \mu >= 200$ amounts to roughly 220 HS06x seconds per event which is at best[5] 11 seconds per event.

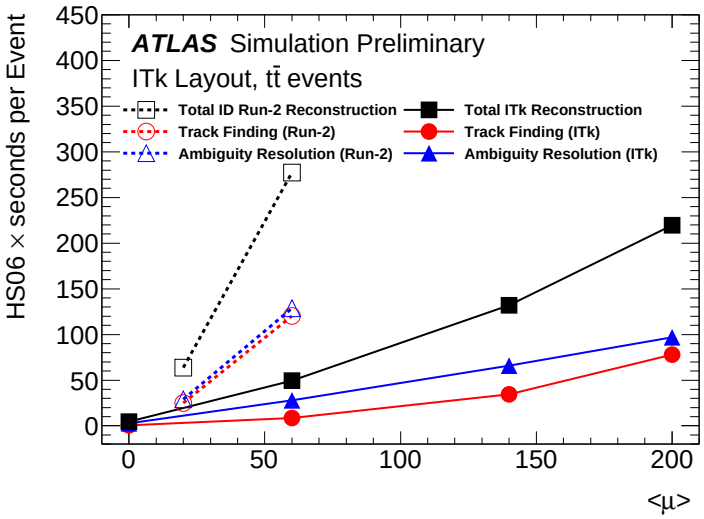


Figure 3.5: Evolution of the CPU resources needed for the Track Finding and Ambiguity Resolution as a function of the pileup. The clear improvement is a consequence of a design layout optimised for technical performances.

Recently, tighter cuts and requirements have been shown to further speed up the total reconstruction pipeline [4]. This comes with a tolerable physics performance compromise, i.e. tighter acceptance cuts mean dropping the less *interesting* particles against CPU improvement. This approach brings the total reconstruction time for ITk to 31.7 HS06X seconds so roughly **1.5 seconds** in the best case (20 HS60 per core). It is important to note that for this number and by design, the ambiguity solver is not used. Instead, the track parameters are estimated within the track finder resulting in less precise estimates.

[5] Considering 20 HS06 per core

Detector	$<\mu>$	Cluster Finding	Space Points	Track Finding	Ambiguity Resolution
ITk layout	200	22	6.5	78	97
Run-2	20	1.5	0.7	23	15

Table 3.2: CPU resources needed for the different steps of the tracking pipeline in ITK and Run2 conditions [4]. Time unit shown is HS06*seconds where modern CPUs have 15-20 HS06 [5] per core.

Bibliography

[1] ATLAS Collaboration, A neural network clustering algorithm for the ATLAS silicon pixel detector, JINST9(2014) P09009, arXiv:1406.7690

[2] ATLAS Collaboration Collaboration Technical Design Report for the ATLAS ITk Pixel Detector,Tech. Rep. ATL-COM-ITK-2018-019, CERN, Geneva, 2018. https://cds.cern.ch/record/2310230.

[3] The ATLAS Collaboration. The Optimization of ATLAS Track Reconstruction in Dense Environments.ATL-PHYS-PUB-2015-006 https://cds.cern.ch/record/2002609/files/ATL-PHYS-PUB-2015-006.pdf

[4] ATLAS Collaboration, Fast Track Reconstruction for HL-LHC, ATL-PHYS-PUB-2019-041

[5] HEPiX - Benchmarking Working Group, lhttp://w3.hepix.org/benchmarking.html

4

Machine Learning

Artificial intelligence (AI) is the concept that allows human made (artificial) entities to act intelligently. The concept definition evolves with our own definition of "intelligence". A sub-field of AI is machine learning (ML) and has an exact and agreed upon definition as "the science that allows computers to take decisions without being explicitly programmed". The later is not an additional challenge set by programmers but rather due to our inability to solve the challenges tackled by ML. Indeed, ML is used e.g. for self driving cars or for image satellite detection because despite our advanced knowledge, such tasks cannot be completely solved by an if-else program. Today, machine learning appears as a revolutionary concept even more than thirty years ago when it was first introduced to the community and that is solely due to the availability of tremendous amounts of data. Machine learning often goes hand in hand with another concept of Big Data, an even stronger relevance to this thesis as the LHC can be regarded as one of the biggest data machines in the world.

This chapter introduces the basic concepts of machine learning with a focus on topics relevant to tracking. The different ML techniques used at the LHC for tracking and beyond will be discussed. At the conclusion of the chapter, an exhaustive table of machine learning techniques suitable for tracking will be presented along with comments on future technologies and trends that ought to be followed.

4.1 Key Concepts and Definitions

The most accepted definition of a learning algorithm is the one by Mitchell [1] "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". This definition contains every key concept used throughout this thesis. In this section, we will define and give examples on the three concepts E,T and P, generally and in the charged particle tracking context.

4.1.1 The Learning Task

Consider the cloud points in Figure 4.1. Let us assume that every point in the figure is a signal emitted by different objects (gas particles, rocks, noise...etc) and that we are only interested in retracing the passage of an alien object, i.e. among these points, a set of approximately 10 points (if linked together with a virtual line) represent the trajectory of an extremely rare and interesting object. Despite our ability to imagine trajectories we have not seen before,

such tasks are too difficult to solve by fixed programs designed by us (human beings). This is especially true since the program spends the vast majority of CPU power in finding and testing trajectories that do not exist (candidates later discarded). Finding the object trajectory is the task and learning the pattern governing the set of points is the mean to achieve the task. A machine learning algorithm is tasked with learning to solve rather than solving this specific example. The learning ability enables the ML to solve even more complex tasks : larger set of points, less intuitive trajectories, and so on.

Figure 4.1: An example of a learning task. Given a set of signals emitted by unknown objects can you identify the trajectory of our object of interest?

An ML task is generally defined by the ideal processing of an example, that is, our available input and desired output. An input is defined as a set of feature vectors that have been measured. An object x that is measured, at multiple instances, might have different properties referred to as features and encoded as dimensions of the object. Thus $x \in R^d$ is an object that has d properties (mass, position , colors...and so on) with n measurements of its properties $x = \{x_i\}_{i\in[0,n]}$. In machine learning and subsequently in this document, we will refer to object properties as features. Features can be discrete or continuous values, images or any structured data that describes the object x. In our earlier example, the object x is the emitted signal and the points represented in the figure are different measurement of the signal x_i. The intuitive features our mind uses in this example are the geometric coordinates of each dot ($x \in R^2$). The collection of all the instance of object x is called a dataset. Generally all the instances of object x have the same dimensionality and in this case, the dataset is a matrix of dimensions $n \times d$. When the object is described by different set of features (images and raw data), the dataset is referred to as non homogeneous dataset. When not every feature of x can be measured at all times, the dataset has missing values. This constraint is very common in machine learning and many solutions are proposed to deal with missing values.

Commonly, the definition of a task precedes and ideally shapes the building of a dataset. This is even more relevant with the imminent rise of active data[1]. Following the definition of a task, many machine learning domains and applications arise. When the task is to assign new instances of an object to known

[1] Agents in Reinforcement Learning that generate/use the data they need to learn from an environment

categories, we refer to the learning task as **classification**. When the instances have to be regrouped under previously unknown categories, the learning task is **clustering**. Because charged particle tracking can be regarded as a clustering of hits, the concept is presented in details in section 4.2. **Anomaly detection** is the task of spotting unusual instances of x in a usually large collection of examples. Another interesting learning task is **density estimation** where the model has to explicitly model the space that examples were drawn from.

Deep neural networks (DNN) are a machine learning technique that is widely used in all of the described learning models. At the core of a DNN structure is **representation learning**. That is the automatic and often implicit learning of abstract features describing the input object. At first representation learning was an inherent property of DNNs but given the profound implications of learning *new* features of data, it stood out as a machine learning field with its own state of the art. Because learning new features from the data (which often simplifies the input) has desirable properties in charged particle tracking (where the input is not simple), we will detail the different techniques of representation learning and metric learning in Section 4.4.

Many more learning tasks exist with their exciting applications. The previous tasks were selected due to their wide usage in high energy physics. Chapter 6 discusses how these algorithms are used in the field.

4.1.2 The Performance Measure

The performance measure is our indication of how reliable a machine learning model is. By definition, the performance is measured on instances of the object that the model has not seen before. This set of instances is called "test set". The exact measure used to evaluate the model depends on the task and domain at hand. We might be interested in the proportion of instances the model predicts correctly, this is referred to as the accuracy. The error rate or loss is the proportion of wrong predictions. In charged particle tracking for example, we are mostly interested in the efficiency of the model and its fake rate. The efficiency is defined as the fraction of correctly retrieved trajectories and the fake rate is the amount of wrong or fake trajectories returned by the model. Moreover, charged particle tracking evaluation requires a significant number of metrics given various physics properties. These metrics are described in Chapter 3. The authors of the TrackML challenge (Chapter 6) created a new performance measure that combines different physics metrics in a single score between 0 and 1. The evaluation of a machine learning model is non trivial as the model might make a lot of sense given certain metrics but not according to others.

Throughout this thesis, different evaluation metrics will be presented and their significance highlighted.

4.1.3 The Experience

The experience is the nature of information that is available to the machine learning model. If the model is provided with only the object features with no further guidance on their meaning, the learning is **unsupervised**. This also implies that no ground truth is available. If we go back to the trajectory example in Figure 4.1, by definition we do not know the object or its trace so we will have to define alternative ways of evaluating the trajectories found by the model. Fields where the problem is grouping data points (objects) with common behavior extensively use unsupervised learning through

clustering. For example, given a massive dataset on customer interaction with a recommendation engine, using clustering allows to find groups of customers with similar interests. Once these groups are defined, it is not feasible to determine whether a customer actually belongs to its assigned group or not. This example generalizes to all the fields where it is impractical to annotate individual objects. In high energy physics, supervised information is produced from simulation making the learning from labels (supervised) a default.

The third type of learning experience is a continuous interaction between the model (called agent) and the data source (called environment). The agent decisions or actions are **reinforced** (positive/ negative reward) through signals from the environment.

In the next section of this chapter, we will detail different types of machine learning relevant to charged particle tracking.

4.2 Clustering

Clustering is the process of creating groups out of unstructured data. Generally, these groups are new knowledge in the sense that the user has no well defined expected structure. In other words, clustering is an unsupervised technique. Therefore, the meaning and usefulness of the found clusters solely depend on the grouping strategy. Strategies of the clustering algorithm are as varied as the algorithm themselves. However, every clustering strategy starts with a *similarity* or *distance* definition. Any algorithm has first to determine the closeness of two points or group of points in order to create larger groups.

As formulated by Backer and Jain [5] "in cluster analysis a group of objects is split up into a number of more or less homogeneous subgroups on the basis of an often subjectively chosen measure of similarity (i.e., chosen subjectively based on its ability to create "interesting" clusters), such that the similarity between objects within a subgroup is larger than the similarity between objects belonging to different subgroups".

4.2.1 Distance and Similarity Measures

A distance is a measure of closeness between two objects. It indicates on some coordinate space, an amount of units to get from a first object A to the second B. In this chapter, we are interested in relative distances. That is, the distance between A and B becomes relevant only at the introduction of a third object C.

A distance on a data set $X = x_0...x_n$ is defined to satisfy the following conditions:

- Symmetry $D(x_i, x_j) = D(x_j, x_i)$
- Positivity $D(x_i, x_j) > 0$ for all x_i and x_j
- Triangle inequality $D(x_i, x_j) < D(x_i, x_k) + D(x_k, x_j)$ for all x_i, x_j and x_k
- Reflexivity $D(x_i, x_j) = 0$ iff $x_i = x_j$

When the two last conditions hold, the distance is also called a metric.
The most popular and intuitive distance is the euclidean distance.

$$d_{euclidean}(A, B) = (A - B)^2$$

The cosine distance measures the cosine of the angle θ between two non zeros vectors A and B.

$$d_{cosine}(A,B) = 1 - \frac{A.B}{|A||B|} = \frac{|A||B|\cos\theta}{|A||B|} = 1 - \cos\theta$$

4.2.2 Hierarchical Clustering

In this section, we will discuss in details the major ideas and techniques of Hierarchical clustering (HC). The specific type of clustering is focused on since it presents an intuitive way of describing tracking. Merging points until a clustering criteria is reached vs merging hits until a particle is fully formed.

The following summarizes the steps taken by any agglomerative clustering.

1. Start with singleton clusters. In particle tracking, each hit starts with its own particle hypothesis. Compute the proximity matrix of the clusters which contains all the pairwise distances.

2. Search the minimal distance

   $$D(C_i, C_j) = \min_{\substack{1<m,l<N \\ m\neq l}} D(C_m, C_l)$$

   where D(*,*) is the distance function of the proximity matrix. C_i and C_j are merged to form a new cluster and N is the number of points in the dataset considered.

3. Update the proximity matrix by computing the distances between the new cluster.

4. Repeat steps 2) and 3) until there is only one large cluster or a stopping criteria is reached

The step 2 is straightforward for clusters comprised of a data point each as a similarity or distance between point is well defined. After the first iteration, the step 2) becomes a distance between a cluster and a data point or between two clusters. There are many possibilities for defining a distance between clusters (also called linkage) and each possibility gives rise to a type of HC. A recurrence formula, proposed by Lance and Williams [2] generalizes the various possible *linkage* scenarios.

$$D(C_l,(C_i,C_j)) = \alpha_i D(C_l,C_i) + \alpha_j D(C_l,C_j) + \beta D(C_i,C_j) + \gamma |D(C_l,C_i) - D(C_l,C_j)|$$

Where D(*,*) is the distance function and α,β and γ are coefficients that take different values depending on the clustering type. As an example, single linkage is a hierarchical clustering that merges clusters choosing the minimal points in each cluster as representative. From the above formula, this configuration takes place at $\alpha_i = \alpha_j = 1/2$, $\beta = 0$ and $\gamma = -1/2$.

4.2.3 Graph Theory Based Clustering

In this section, we choose to present one clustering algorithm that uses the following relevant features : relies on a similarity graph, uses dynamic merging criteria and several points are considered for cluster merging. Some of these points make Chameleon [3] one of the only clustering algorithms able to find arbitrary shaped and variably dense clusters.

Chameleon as its name indicates, is a dynamic hierarchical clustering algorithm that uses more than one criterion to merge data points into clusters.

It is based on building a k-nearest-neighbor graph where an edge represents the similarity between data points (nodes). The graph is then divided into sub-graphs with only enough nodes to compute the similarity: The edges are progressively eliminated if both vertices (nodes) are not within the closest points related to each other. The last step is the merging of the sub-graphs. This is summarized in Figure 4.2.

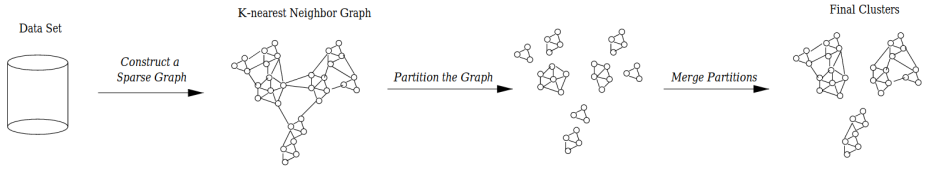


Figure 4.2: Chameleon framework. Source: Karypis et al. [3]

The key feature of this algorithm is that it determines the pair of most similar sub-clusters to merge by taking into account both the inter-connectivity as well as the closeness of the clusters overcoming the limitations resulting from using only one of them (most clustering algorithms).

Clusters closeness, of two clusters C_i and C_j is measured by computing the average similarity between the points in C_i that are connected to points in C_j. In graph words, that is the average weight of the edges connecting vertices in C_i to vertices in C_j. This metric choice makes the algorithm more tolerant to outliers. The second merging criteria is the inter-connectivity. The absolute inter-connectivity between a pair of clusters C_i and C_j is defined to be the sum of the weight of the edges that connect vertices in C_i to vertices in C_j. Both criteria are normalized by their internal closeness and inter-connectivity.

Despite an established clustering efficiency, Chameleon suffers from its high computational complexity. Several works combined parts of Chameleon with faster algorithms or parrallelized its stages to take advantage of its high accuracy without computational cost. Li et al. [4] combined Chameleon with Clustering Feature Tree[2] (CFT) reducing the running time of Chameleon by a factor of 10. Unfortunately, the performances are still not realistic for our use case with a run time of 200 seconds for a dataset of 6K points.

Clustering algorithms use the data provided by the user and their results at best reflect how well the different data dimensions describe the patterns. But what if the dataset at hand is complex and has hidden properties not directly available as features? Deep learning is the art of building abstractions from the data in order to reach a better representation. In the next section, we describe the different notions central to deep learning with the most relevant models.

4.3 Deep Learning

At the heart of deep learning lies the concept of neural networks. Neural networks are the brain inspiration models that scientists came up with to emulate intelligence. Despite our limited understanding of the brain and its neurons, artificial neural networks in their basic imitation of information processing are at the heart of most if not all major successes of AI. Following our imagination of the brain, neural networks are layers of small processing units called neurons, through which a signal is passed and given the contribution of the

[2] A height-balanced tree data structure with the linear and square sum of data points representing the clustering features

current neuron (its function), the signal value is updated. At the end of the process, the output of Artificial Neural Networks (ANNs) are not thoughts but numerical values. The reason of ANNs success is their ability to learn non linear mappings. ANNs are often refereed to as universal approximators because they learn a function $\mathbf{f}(x)$ that maps input x to output y by learning a parameter θ. The prediction of the network is referred to as y'_i and is therefore $y'_i = f(x) = x.\theta$ with θ known as the weights of the network.
The most common type of ANNs are **Feedforward networks** in which the information flows from input to output with no feedback connections. ANNs with feedback connections are called **Recurrent Networks** and are widely used in time sequence modelling.

Figure 4.3 shows the basic principle behind ANNs. An input vector describing the features (4 dimensional in the example) of the object x is passed as input. Each feature is connected to the next layer, referred to as hidden layer as it stands between the input and output. The input vector is mapped to the latent variables h through the connection weights w. When the layer nodes are fully connected, i.e. each node is linked to the next one by a weight, the network is called **dense**. The function $g(x)$ is defined per layer and is called the activation function. Common forms of $g(x)$ are:

1. Sigmoid $g(x) = \frac{1}{1+e^{-x}}$ with values between 0 and 1
2. Relu Rectified linear unit $g(x) = max(0, x)$
3. Tanh Hyperbolic tangent activation function $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

In supervised machine learning, the goal of the network is to reduce the error or loss between the desired output (target) and the predicted output (prediction). The loss is therefore a form of difference between output and target. A popular measure of the loss is to use the Mean Square Error (MSE) when the output and predictions are of continuous form. MSE is the sum of squared distances between the target and the predicted values :

$$Loss_{MSE} = \frac{\sum_{i=1}^{n}(y_i - y'_i)^2}{n}$$

When the task is a binary classification, a common loss formula is the binary cross entropy

$$Loss_{BinaryCrossEntropy} = \sum_{i=1}^{n} y_i \log y'_i + (1 - y_i) log(1 - y'_i)$$

The choice of the activation function, of the number of hidden units and the number layers, the optimizer, the loss function, the batch size and the learning rate are the network hyper parameters and their tuning is an essential step in neural network based machine learning. When the network has more than 2 hidden layers, it is referred to as deep neural network.

In this chapter, we choose to describe in detail the architecture of two popular Neural Network models due to their increasing popularity in high energy physics.

4.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are often presented as the models through which machine learning sees the world. In seeing, we first refer to images or image sequences (videos). CNNs take as input images and are able

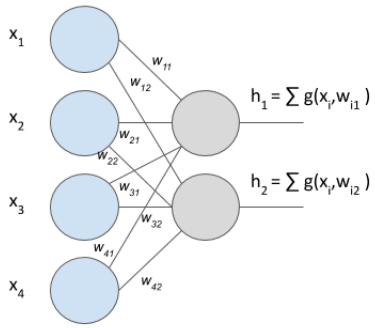


Figure 4.3: The principal behind neural networks. In the example, x is the input with its four dimensions (features).

to give weights to each pixel in the each commonly detecting patterns or identifying objects. A CNN, unlike any other network, extracts features from raw images. With enough layers of abstraction, the features are mapped to patterns and the patterns to complex objects. Contrary to a dense network that operates on a value by value basis, a CNN is able to extract complex features from an image by using filters. These filters allow the network to detect spatial and temporal dependencies in the images.

Figure 4.4 shows an example of a CNN architecture to identify the hand written digits images. The input is a 28x28 image that goes through 10 convolution layers with a 5x5 kernel. The kernel is a usually smaller matrix that scans the successive images by convolutions. The different convolution layers are responsible for extracting the high level features such as the edges, colors and orientation. Each added layer extracts finer characteristics and this is why complex images often require many more convolution layers to extract the relevant patterns.

Pooling layers are then used to primarily reduce the dimensionality of the image to reduce the computation powered required for processing. Using Max pooling as shown in the pipeline of Figure 4.4, will return the maximum value in the kernel. This is useful to suppress the noise present in images. Another alternative is to use Average pooling which performs the average of the kernel used to scan the image. The choice of such parameters strongly depends on the data and problem at hand.

In the example discussed, we can see that after two sets of Convolution and pooling layers, the processed images are transformed to a 4x4 matrix. The features extracted by the CNN are therefore present in this last 4x4 image. To perform classification, dense layers are added. In the example, exactly 2 fully connected layers are used to extract non linearity from the 4x4 image and convert it into a 10 dimensional probability output vector.i.e. a probability for each of the ten digits classes.
The network architecture presented so far is large and complex. When running on a relatively small dataset, it will learn to extract exactly the features seen at training with no capacity to generalize to further examples. This is referred to as overfitting. In machine learning, the design of network architecture is

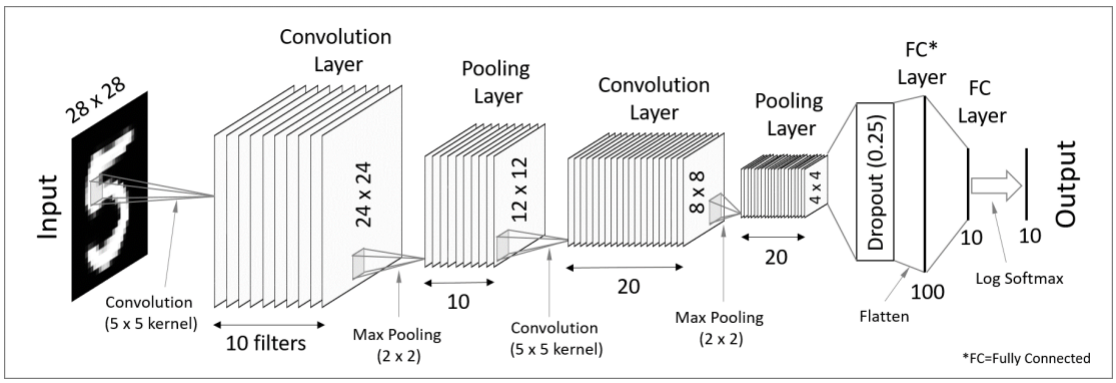


Figure 4.4: Convolutional Neural Network used to classify the MNIST digits dataset. Image source : Code to Light tutorial

dictated by a need to run on real world problems thus making generalization to completely unseen images during the training. The dropout concept was introduced to solve exactly these shortcomings. It implies randomly ignoring a number of nodes at each iteration. This has the effect of simulating different architectures every time thus preventing the nodes to overfit when seeing each time the same examples and adjusting their weights to the correct combination.

4.3.2 Long Short Term Memory Neural Networks

Recurrent neural networks (RNNs) are designed to recognize patterns in sequences of data using a temporal dimension. RNNs in general are particularly relevant in speech recognition problems where to understand the meaning of a sentence, the model has to understand its relationship with prior statements. Long Short Term Memory Neural Networks (LSTM) are a type of RNNs that improves the basic idea of memory to long term memory or context.

Specifically, LSTMs store past information dynamically. The higher the weight of the information, the longer it is stored in the node. This is achieved thanks to the concept of cell state : a flow through the entire network chain with very few changes to it. Information is let in the nodes using gates in the form of sigmoid functions [6].

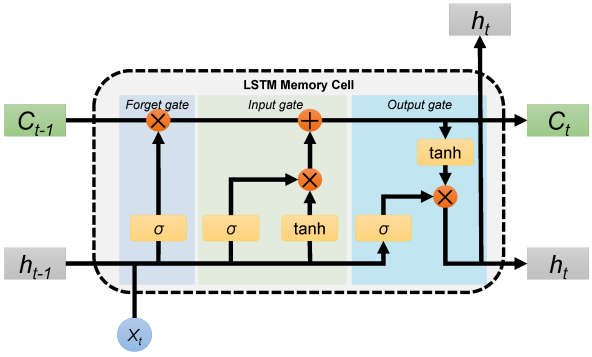


Figure 4.5: LSTMs cell structure. The state is the C_t channel that crosses the cell. In this example, the h_t information is computed and stored (hence the double appearance).

A special type of LSTM that is relevant to charged particle tracking is

the Sequence to Sequence LSTM. It appeared first for sentence modeling and translation. An encoder RNN network maps the input sequence (text or images) to a vector representation that is used as initial state to a decoder RNN. The latter generates the target sequence. When the sequence is an image, CNNs are used in the encoding step.

Depending on the size of the input and output, several architectures are possible:

- One to many. Such as generating caption for an input image
- Many to one. Such as text classification, common to sentiment analysis.
- Many to many. Such as text translation or video labelling.

4.4 Metric Learning

In section 4.2.1 we defined the distance concept with its commonly used examples. In this section we introduce the notion of metric (distance) learning as the machine learning field interested in learning from the data domain related distances. Metric learning is used in conjunction with techniques that reply on a similarity computation such as clustering algorithms, and nearest neighbors approaches.

Metric learning supposes the presence of similarities in a given dataset. More specifically, there exist pairs or groups of objects in the dataset that have a higher similarity (affinity) among themselves compared to any other object in the dataset. Formally, let us introduce S as the set of data points with high similarity between them.

$$S = \{x_i...x_k\} \text{ with } x_i..x_k \in X \text{ and } x_i...x_k \text{ share } \textbf{similar} \text{ properties}$$

Each object in S has non similar properties with all the objects outside S. This naturally defines the set D as :

$$D = \{x_j...x_n\} \text{ with } x_j..x_n \in X \text{ and } x_j...x_n \text{ do not share } \textbf{similar} \text{ properties}$$

Generally the concept of similarity is defined relatively to a domain. In face recognition, similar images are defined as different instances of the same person. In recommendation engines, a major goal is to classify clients in groups with similar purchase behaviour. In object tracking, traces left by the same entity are considered similar and specifically in charged particle tracking, our ensemble S contains points produced by the same particle. A collision event can be regarded as multiple S ensembles with each one representing properties from a single particle (hits).

Metric learning aims at learning a metric or distance **f()** such that:

$$\forall (x_i, x_k) \in S \text{ and } (x_j, x_n) \in D, f(x_i, x_k) < f(x_j, x_n)$$

This means that the learned metric will produce smaller distance values for objects with similarities compared to object that do not share properties. Such metric is often defied for pairs or triplet objects. Figure 4.6 shows an example mapping of different particle trajectories in the transverse view of the TrackML detector (detailed in Chapter 6). The learned metric is applied to the dataset projecting the 2D points in a new space where smaller distances between similar objects can be observed.

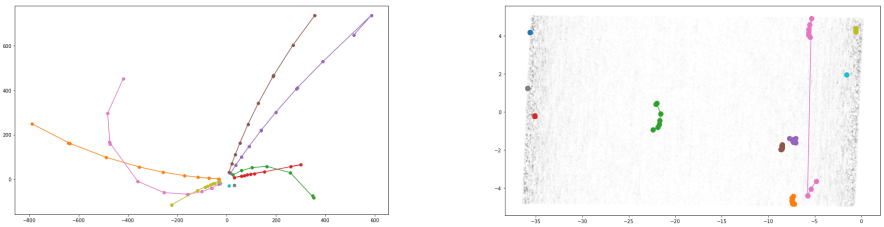


Figure 4.6: Illustration of metric learning applied to a charged particle tracking. (a) Tracks are color coded with the hits connected by continuous lines. (b) Each track is projected into a learned space, color coding is similar to (b). The tracks are generally shown to cluster in small regions of the new space.

Metric learning is generally applied prior to a clustering algorithm that uses an euclidean distance on the learned space. As shown in Figure 4.6, a KMeans can be applied on the learned space to correctly cluster the tracks. Depending on the metric learning algorithm, the original dataset can be mapped globally to a new space or rather individual pairs/triplets receive a *similarity score*.

In the remainder of this section, we present different techniques for metric learning. We choose to describe only non-linear techniques as we believe charged particle tracking cannot be efficiently solved with linear mappings.

4.4.1 Deep Learning Based Techniques

The work of Bromley et al. [7] pioneered non linear metric learning with the use of a convolutional neural networks (CNN) to map images into a space where a semantic distance can be approximated. Specifically, if the mapping learned by the CNN is represented by $G_W(X)$ with X being the image in input and W the weights that parameterize the network, then the network learns the norm $||G_W(X) - G_W(X')||$ as smaller for positive image pairs (same person) than for negative pairs (different person). Figure 4.7 describes the architecture of the model famously known as *Siamese* architecture. A Siamese architecture was first proposed in 1992 for signature verification and it is composed by two subnetworks sharing the same parameters. The weights W are shared between the network to extract similar properties and the output E_W is a form of energy between pairs, low for similar ones and high for different ones.

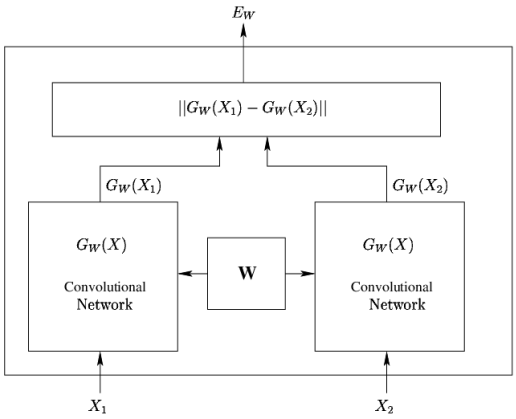


Figure 4.7: The Network architecture used by [7] to learn a similarity metric.

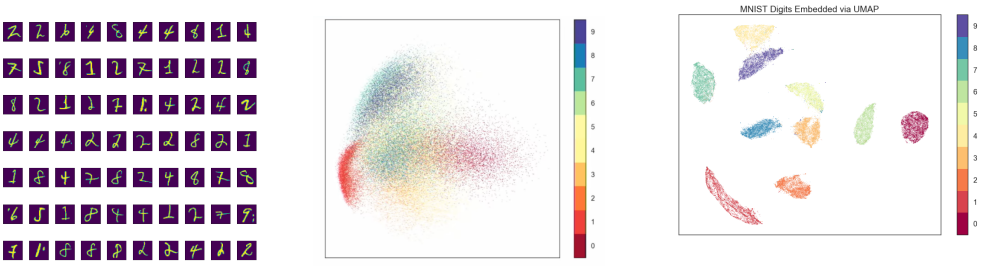
Although the Siamese architecture is considered the first deep learning based metric learning model, different architectures and model constraints were proposed to solve specific image related challenges. The authors in [8]

trained a network discriminatively for face verification while Chechik et al. [9] learned a ranking function using triplet loss. Qian et al. [10] used precomputed activation features and proposed the learning of feature embedding via distance metric for classification.

4.4.2 Uniform Manifold Approximation and Projection for Dimension Reduction

Dimensionality reduction is another way of projecting a complex dataset into a simpler one where the different classes are more visible and often better separated. Figure 4.8 shows an example of the projection of the hand written digits dataset (4.8(a)) into a 2D embedding space where the different number classes are well separated (4.8(c)). An image from this dataset is a 792 dimensional vector and each vector is contracted to only to 2 dimensions in this example.

Techniques used for dimensionality reduction can be grouped into two large families : Matrix factorization and Neighbor graphs. Principal component Analysis (PCA) is one of the most popular matrix factorization techniques. It is widely used in machine learning especially as data processing technique but fails at finding the local structure of the data and therefore at providing a well separated output. On the other hand, Neighbor graphs techniques rely mostly on the building of local relationships within the classes and generally provide well separated classes in output. Our goal being the efficient clustering of particle hits, we will focus on neighbor graphs and particularly on Uniform Manifold Approximation and Projection (UMAP) [11].



(a) Digits dataset images.

(b) PCA projection

(c) UMAP embedding of the Digits dataset.

Figure 4.8: Example of the application of dimensionality reduction technique UMAP on the popular image dataset Digits. Each image in the dataset is represented as a 2D point in the learned data manifold presented in 4.8(c). The different number classes cluster nicely in the 2D manifold.

UMAP is the state of the art technique for dimensionality reduction and manifold approximation. Despite the algorithm being relatively new (2018) it has had a profound impact in the dimensionality reduction world. The two main factors for this impact are the scalability of the technique on high dimensional datasets and the possibility of projecting new samples without refitting the embedding each time.

UMAP constructs a high dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible. Effectively, data points in the graph are connected using a local connection radius, i.e. points are connected if these radii overlap. This unique property helps the algorithm in preserving the data structure as demonstrated in Figure

4.8. The number of neighbors for each point is a hyper parameter to optimize in the algorithm.

Bibliography

[1] Mitchell, Tom. (1997). Machine Learning. McGraw Hill. p. 2. ISBN 0-07-042807-7

[2] Lance, G.N., Williams, W.T.: A general theory of classificatory sorting strategiesii. clustering systems. The computer journal 10(3), 271–277 (1967)

[3] Karypis, George, Eui-Hong Han, and Vipin Kumar. "Chameleon: Hierarchical clustering using dynamic modeling." Computer 32.8 (1999): 68-75.

[4] Li, Jinfeng, Kanliang Wang, and Lida Xu. "Chameleon based on clustering feature tree and its application in customer segmentation." Annals of Operations Research 168.1 (2009): 225-245.

[5] E. Backer and A. Jain,"A clustering performance measure based onfuzzy set decomposition,"IEEE Trans. Pattern Anal. Mach. Intell., vol.PAMI-3, no. 1, pp. 66–75, Jan. 1981

[6] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[7] J. Bromley, I. Guyon, Y. Lecun, E. SŁckinger, and R. Shah.Signature verification using a "siamese" time delay neuralnetwork. InNIPS, 1994

[8] Chopra, R. Hadsell, and Y. LeCun. Learning a similaritymetric discriminatively, with application to face verification.InCVPR, June 2005.

[9] Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scaleonline learning of image similarity through ranking.JMLR,11, 2010

[10] Q. Qian, R. Jin, S. Zhu, and Y. Lin. Fine-grained visual cat-egorization via multi-stage metric learning. InCVPR, 2015

[11] McInnes, Leland, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction." arXiv preprint arXiv:1802.03426 (2018).

5

Approximate Nearest Neighbors

The notion of similarity was recurrent throughout this document. Firstly in clustering as the notion around which to group data points and later in metric learning. If the relationship between objects cannot be described by simple distances (similarities), it becomes possible to learn one from the data. If we look into more details at the examples discussed in the metric learning section, we see that learning similarities between objects is only the first in tasks such as person identification, object tracking or document analysis. For example, given an image of a person, its identity is retrieved by finding the most similar person in the database. Such databases, whether of people, objects, songs or movies are generally very large (up to a billion stored records) for giant companies such as Google or Facebook.

The challenge is then formulated as : "Given a similarity measure and an object Q, how do we efficiently retrieve the most similar objects to this reference object Q". This is an old and notorious problem in computer science known as Nearest Neighbor search. If we are willing to trade off some accuracy for speed and accept an approximate solution, the problem becomes known as Approximate Nearest Neighbor (ANN) search.

Approximate Nearest Neighbor search applications are diverse and numerous in the age of Big Data and recommendation engines. Famous examples include music recommendation at Spotify and Facebook AI similarity search engine (images).

The performance measure of exact nearest neighbor search is straightforward. The goal is to find the exact elements, the fastest. The evaluation is therefore one dimensional : speed. In ANNs, the precision notion is used instead of the accuracy. The precision is the fraction of correct elements that are retrieved by the total retrieved elements. Different techniques are generally evaluated according to these 2 dimensional metrics.

Figure 5.1 illustrates the state of the art in ANN search as of 2019 with a comparison of the two fastest techniques : Facebook AI similarity search (Faiss) and Navigating Spreading-out Graphs (NSG) by the Alibaba group.

The two fastest techniques are compared on a 100 million vector dataset DEEP1B [2] where each data vector has 96 dimensions[1]. The best speed performance is slightly above 10K queries per second. According to the authors of the study [4], the best performing technique NSG-16core is the random splitting of the 100 million dataset across 16 cores and the use of simultaneous queries with a merging of the results. Faiss-16core and Faiss-GPU on the

[1] Although the dataset size is 1 billion vectors, only a subset of a 100 million was used for the study due to RAM limitations.

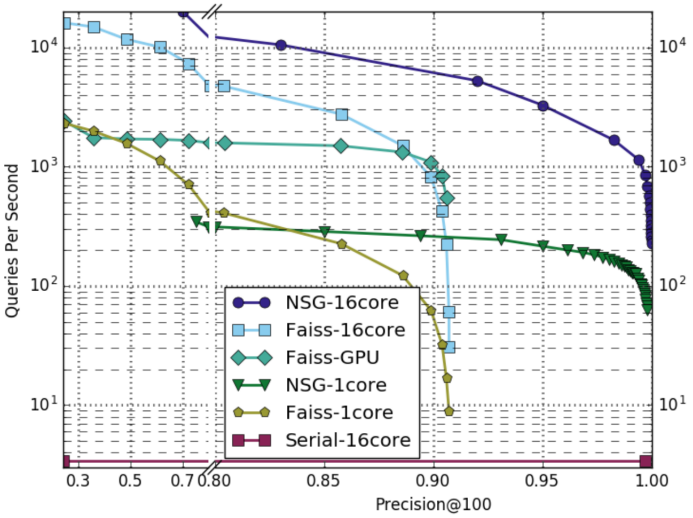


Figure 5.1: Latest ANN performance comparison on 100 million vectors dataset. Source : Fu et al [4]

other hand process the 100 million vectors in parallel without any split. The remainder of this chapter is structured as follows. In the next section a formal definition of the ANN problem is presented, followed by a detailed description of the different ANN existing approaches and their applications. Lastly, we present the link between ANNs and charged particle tracking.

5.1 Problem Definition

Given a d dimensional dataset $X = \{x_1, x_2, ..., x_N\}$ where $x \in R^d$ and a query item $q \in X$, the nearest neighbor (NN) of q in X is defined as :

$$NN(q) = \text{argmin}_{x \in X} D(q, x)$$

where $D(,)$ is the chosen distance function. The approximate nearest neighbor (ANN) of q in X is the item x where:

$$D(q, x) < (1 + \epsilon) D(q, x^*)$$

x^* being the true nearest neighbor of q and ϵ a very small parameter. Figure 5.2 describes the necessary steps to get from a query point $q \in X$ to the set of k closest neighbors using an ANN index.

5.2 Similarity Search Models

Different real world applications and constraints have called for different ANN index structures and search algorithms. There are tree-structure based approaches, hashing-based approaches, quantization-based approaches and graph-based approaches. It is often accepted that graph based techniques have yield superior results over the years in term of precision and queries throughput. Despite this fact, many different aspects are to be considered when choosing an ANN technique and this explains why major companies with search challenges are not necessarily using graph based techniques. Such factors include index memory size, index building time, GPU portability, parallel queries and storage format.

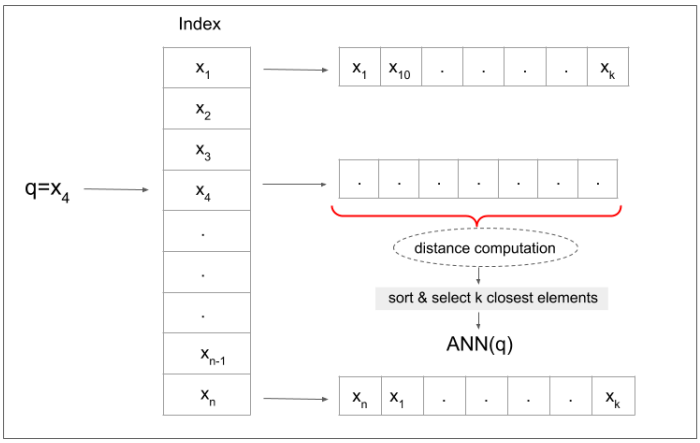


Figure 5.2: Approximate Nearest Neighbor search. Distance to the query point q is only computed on a small subset of the dataset. The candidate list returned ANN(q) are the k sorted nearest neighbors to the query.

In this section, the different ANN techniques will be detailed and discussed along these technical considerations.

5.2.1 Tree Based Techniques

Most tree based ANN techniques are built around the concept of a K-dimensional tree (KD-tree). A KD-tree is a balanced binary tree where the dataset is recursively split into two halves until reaching the leaf level where each leaf is a data point. The split occurs using a hyper-plane orthogonal to a chosen dimension at a threshold value. When searching for the neighbors of a query point, the nodes are searched in the order of their distance to the query point. In approximate nearest neighbors, the search process is terminated after a specific number of distances computations. Depending on the specific technique used, more than one tree is built and searched.

Annoy is a popular tree based ANN techniques library [1] that has the particularity of using static files data structure mapped into memory and shared among multiple processes. It is used in Spotify for music recommendations by searching similar users/items in high-dimensional spaces and large datasets (many millions according to [1]).

Annoy uses random projections to build a tree. Hyperplanes are used to divide that dataset recursively into two subsets. As opposed to the hyperplane choice in KD-tree, Annoy samples randomly two points from the subset and builds the equidistant hyperplane. Multiple trees are built in parallel to increase accuracy through a user chosen parameter. The more trees are used, the slower the queries become.

The authors of Annoy published an ANN benchmark [3][2] that is used and referred to in the community as the principle source of comparison between different ANN techniques. The different aspects investigated in the benchmark are :

1. The recall, i.e. the ratio of true nearest neighbors returned by a query
2. The size of the data structure (index) in Kilobyte (kB)
3. The time it takes to build the index in seconds

[2]First version appeared in 2018

4. The number of distance computation

5. The time it takes to perform a query or the throughput (number of queries per second)

Among these measures, we are particularly interested in the index build time and the throughput. These are determining factors in charged particle tracking.

The benchmark compares the different performances of tree based ANN techniques as well as several graph based and random projection based techniques. One of the conclusions of the authors is that generally graph based ANN techniques are found to perform the best. However, the time to build the data structure (index) is found to be much larger than tree based or inverted-file based algorithms. In the two next sections, we will present in details the concepts behind graph based ANN techniques and Inverted file based techniques and conclude with a technical comparison of these techniques.

5.3 Graph Based Techniques

A graph is a data structure. A set of nodes or vertices connected by edges. When the nodes represent data points and the edges a form of similarity between points, we refer to such graph as "Proximity graphs". Proximity graphs impose constraints on the edges to solve the ANN problem. In the graphs setting, given a node q, the core idea is to find the shortest path that traverses the k true neighbors of q. Graph algorithms differ mainly in the way they traverse such graphs. One of the first proximity graphs is known as the Delaunay graph where the edge constraint enforced is that from any node p to any node q, there exists a path. Such constraint guaranties the finding of the true neighbor (it will be traversed) but can lead to a fully connected graph in high dimensions making the approach computationally limited. Another example of edge constraints is to limit the number of edges per vertex thus shifting the problem to an approximate nearest neighbor one (the true neighbor is not guarantied to be visited). The K-nearest neighbour graph imposes such constraints. Fast Approximate Nearest Neighbour Graphs (FANNG), proposes an efficient data structure to reduce search time. At the query time, only *relevant* edges are considered to build a minimal graph. When a node *p1* is connected to a node *p2* then all the connections from *p1* to any other node *p3* that is closer to *p2* than *p1* are discarded (irrelevant in the search).

Other approaches such as the Hierarchical Navigable Small World (Hnsw) [5] propose a layered data structure where vertices are associated with different "levels" of search. Hnsw is to date the fastest ANN graph based approach with a logarithmic search time[3]. Hnsw are based on the small world type of graphs that first appeared in the context of the Milgram experiment. The experiment principle was that participants had to convey information using a network of people they did not know. Contrary to common intuition, the number of hops (intermediary people) was as low as 5 persons, i.e. for the information message to pass from any two people randomly chosen in the network, it needed only 5 links (and thus was born the expression "small world"). Formally, small worlds graphs are graphs with the constraints that most nodes are not connected but the neighbors of any given node are likely to be neighbors of each other and that there exists a small number of hops between any

[3]NGS [4] authors, discussed in this section, claim faster query time but we did not test their implementation for this study.

two nodes. Few decades later Kleinberg added to the small world graphs the navigation property enabling efficient search in these structures with polylogarithmic scaling of the greedy algorithm. Hnsw reduced even further the search complexity to logarithmic by using hierarchies in the navigable small world graph.

The innovative idea in Hnsw is to spread edges in hierarchies according to their length scale (a similarity property). This results in a fixed number of nodes to explore (independently on the dataset size) with the search algorithm "jumping" from a hierarchy to the other.

Figure 5.3 describes the layered approach of Hnsw. The search starts from large radius layers (layer 2 in the Figure) until a local minima is met. From a given point, the algorithm tries to converge as fast as possible toward the query point (shown in green in Figure 5.3). In the example shown, the search only takes 3 hops (edges).

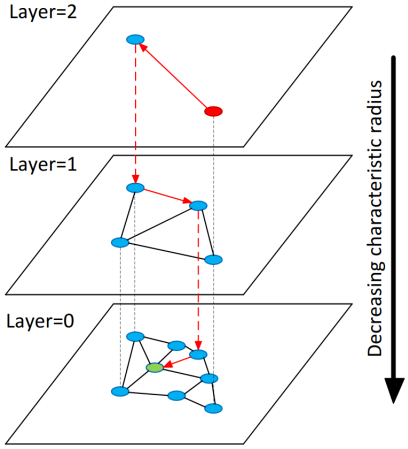


Figure 5.3: Illustration of the search in hierarchies of Hnsw. Source :[5]

The data structure is therefore a multi-graph spread across layers. It is possible to speculate around the meaning of such a structure with respect to the data being represented. In a sense, the hierarchies represent an interesting "clustering" of the data according to some radius. An attempt at uncovering these properties of the structure and its implications in charged particle tracking is discussed [6]. A contribution of this work is the structuring of particle events in such structures and the finding of a new way to perform tracking. The major (and probably only) drawback in Hnsw is the index building time which makes it impractical for large scale datasets.

The navigating spreading out graph (NSG) is proposed to address exactly this issue. The algorithm targets especially large scale datasets with up to a billion (high dimensional) data points. More specifically, the authors implemented the algorithm to target the Taobao (Alibaba) e-commercial needs in terms of user commodities recommendation (128 dimensional dataset) with daily updates. NSG was tested on a 45 million dataset with a distributed search procedure that split the data into 12 subsets building 12 different NSG and merging the results. The authors found it impractical to build a single graph for the billion scale dataset and used a distributed search to reach the platform requirements (<10ms per query at 98% precision). The authors claim that NSG presented improvements of 5-10 times over the previously used technique, a Product Quantization With Inverted File Index (IVFPQ) technique by Facebook AI Similarity Search (Faiss). ANN techniques proposed by Faiss are detailed in the next section.

NSG [4] comes with the following four design principles : connectivity

of the graph, lower average out-degree (number of edges of a node), shorter search paths and especially a reduction of the index size. To achieve this, the authors propose a new graph structure referred to as Monotonic Relative Neighborhood Graph (MRNG). Essentially, the authors ensure that the graph is monotonic but sparse enough to allow an efficient build time. A monotonic graph means that given any two nodes in it, it is possible to find at least one monotonic path. That is a path that continuously shortens the distance between the two nodes. This means that the resulting graph has more connections (ensuring faster retrieval of neighbors) but also more efficient in building those additional edges. Technically this is achieved by always starting from a node that is known to lead to a monotonic path and called Navigating Node. NSG also applies a cut on the maximum number of outgoing edges from any node (a fixed number). This results in the loss of hubs structure in the graph which is not relevant to our use case.

Table 5.1 summarizes the major differences between the two ANN graph based techniques Hnsw and NSG. AOD refers to the average out degree or number of links from a node. NN% represents the percentage of the nodes which are linked to their nearest neighbor. These results were reported by the authors of NSG [4] on running both techniques on the SIFT1M dataset that has 1 million data points of 128 dimensions. According to the same source, NSG is slightly faster than Hnsw for the same precision (3.10^3 queries per second).

	Build time (s)	Index size (MB)	AOD	NN%
Hnsw	376	451	32.1	66.3
NSG	274	153	25.9	99.3

Table 5.1: Summary of the differences between Hnsw and NSG compared on a 1 million size dataset with 128 dimensions

5.4 Facebook AI Similarity Search

Faiss is the similarity search library developed by Facebook AI. The novelty in Faiss is the offloading of computations onto the GPU. As a result, although Faiss is not faster than Hnsw or Annoy on CPU, it is massively faster on GPU. Similarity Search with Faiss relies on:

- An Inverted File (IVF) index : A two level tree structure obtained by running the KMeans algorithm on the dataset. The IVF links every data point to its representative: the cluster centroid.
- Asymmetric Distance Computation (ADC) : The distance computation takes place between the query point and an encoding of the original data point (compressed).

These two concepts allow the parallel processing of ANN queries (batch mode) as well as the handling of very large datasets (due to the compression). The important performance gain with Faiss on GPU is demonstrated in Chapter 7.

5.5 Relevance to Charged Particle Tracking

It is now clear that when searching for similarities in a dataset, ANNs are the fastest alternative. We propose to use ANNs for particle tracking since

they allow a **fast access to data points sharing similar properties**. The definition of "similar" is specific to the task (and data) at hand. Using the analogy of person identification that often relies on a CNN to decide whether two images represent the same person, we propose to augment the ANN approach with a model that decides whether two hits belong to the same particle. Figure 5.4 illustrates the result of an ANN query in the context of charged particle tracking.

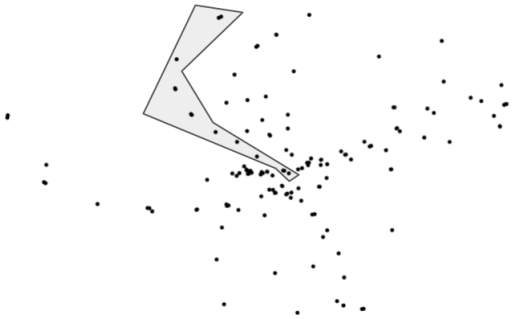


Figure 5.4: ANN applied to charged particle tracking. A bucket is a track approximation

Bibliography

[1] https://github.com/spotify/annoy

[2] A. Babenko and V. Lempitsky. Efficient indexing ofbillion-scale datasets of deep descriptors.Proceedingsof the IEEE Conference on Computer Vision andPattern Recognition, pages 2055–2063, 2016.

[3] Aumüller, Martin, Erik Bernhardsson, and Alexander Faithfull. "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms." Information Systems 87 (2020): 101374.

[4] Fu, Cong, et al. "Fast approximate nearest neighbor search with the navigating spreading-out graph."Proceedings of the VLDB Endowment 12.5 (2019): 461-474

[5] Malkov, Yury A., and Dmitry A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." IEEE transactions on pattern analysis and machine intelligence (2018).

[6] Amrouche, Sabrina, et al. "similarity search for charged particle tracking." 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019.

6

The Tracking Machine Learning Challenge

6.1 Introduction

Although the TrackML challenge [1][2] is not the first public machine learning challenge addressing a high energy physics problem, it is the first that encapsulates all the complexity of a HEP problem without falling into any specific ML category. This is particularly interesting from a machine learning perspective as generally machine learning basics state that a problem is either supervised or unsupervised, a classification or regression. What about charged particle tracking for which we have labels from simulation but no classes definition[1]?

In this chapter, we will take a top-down approach first discussing machine learning approaches used in HEP and the general directions of the field (regardless of the experiment). Finally, the TrackML challenge will be detailed as it represented the author's qualification task as well as the initial dataset on which the proposed ML models were designed.

6.2 Machine learning for High Energy Physics

In charged particle reconstruction, the multitude of experimental apparatuses has led to diverse tracking techniques. However, no standard technique achieves general track finding under universal experimental conditions. With tracking generally evolving towards automatic and parameter-free techniques (see Figure 6.1), an interesting goal is to overcome the architecture dependence. Machine learning contrary to classical combinatorics provides the ability to learn and improve from experience. It is therefore able to generalize to unseen examples and to learn implicit patterns. In contrast, combinatorics and classical pattern recognition techniques do not evolve with experience and then are not robust against changing conditions.

The work from Peterson [3] is the earliest attempt at using machine learning for charged particle tracking. He proposed the use of Neural Networks for track finding in 1989. Every hit constitutes a node in a very large neural network with an update rule favoring adjacent segments with small angles between them and discarding track bifurcations. The network showed interesting results on small datasets (state of the art in 1989). His approach was enhanced and implemented 13 years later with the Cellular automaton in the Hera-B experiment increasing the weights of the neurons to find additional points [4].

[1] Particle cannot be represented as classes otherwise only a unique example (instance) would be available per class.

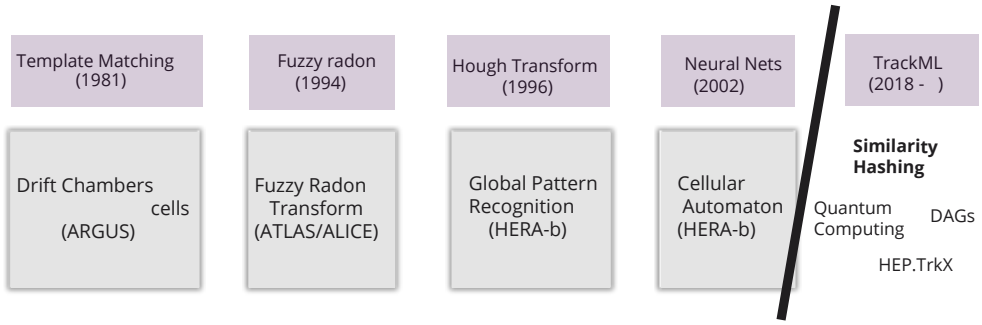


Figure 6.1: Evolution of tracking techniques across HEP experiments. Within the time frame of this PhD thesis more (including ML based) tracking techniques have been proposed than during the last 40 years. The list of techniques is not exhaustive.

Since that period and until very recently (2016), the use of machine learning in HEP summed up to Boosted Decision Trees (BDTs) for classification tasks.

Starting from 2016 on the other hand, a number of R&D projects evolved around the use of *modern* ML techniques for challenging problems such as tracking, vertexing, trigger or jet grooming. In particle tracking, a community conference/workshop annually gathers experts from different experiments and backgrounds to discuss novel software and hardware techniques : Connecting the Dots (CTD) [5] is at its 7th edition. Figure 6.2 shows the constant increase in ML related contributions over the years.

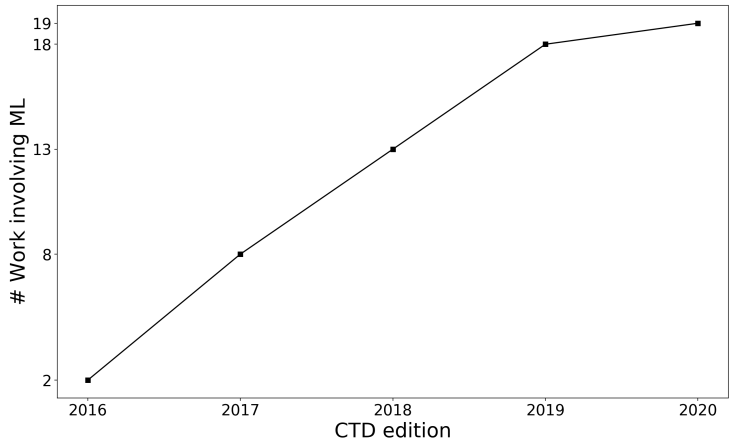


Figure 6.2: Number of contributions based on machine learning (ML in the contribution title) over the last 5 years. A CTD conference has roughly 30 oral contributions. Different components of this thesis were presented in each CTD conference starting from CTD 2017.

The last bloc of ML based techniques showed in Figure 6.1 are amongst the many discussed at CTD conferences and sometimes their evolution can be tracked over the years. An important number of contributions such as quantum computing for tracking [6], graph neural networks [1], HEP.TrkX [7] and similarity search (proposed approach) were built based on the TrackML challenge dataset released in 2018. Before diving into the challenge motivation and detector layout, we will briefly review the major ML based tracking

techniques :

- The multi-Threaded Directed Acyclic graph (DAG) [1] got the third place in the second phase of the TrackML competition with a score of 92.1% in 7.28 seconds[2]. The approach is based on DL models that predict doublet and triplet compatibility inside a DAG that is split on different threads.

- The HEP.TrkX [7] was a project led by 12 researchers from three universities with the aim to identify and develop cross-experiment solutions based on machine learning algorithms for track reconstruction. The authors proposed a graph neural network (GNN) model for tracking showing results on a subset of the TrackML dataset (roughly 2k particles). A previous study showed performance of RNNs and LSTMs.

- Exa.TrkX is a followup of HEP.TrkX to carry on exploring HEP advanced tracking algorithms. This time the project focuses mainly on GNNs and their deployment on FPGAs. The Exa.TrkX groups seven universities/institutes with 25 researchers. The latest results showed performances of a GNN pipeline on (still) a subset of the TrackML dataset.

- A Quantum algorithm was built on top of the GNN proposed by Exa.TrkX to perform track reconstruction [10]. The study showed the feasibility of such alternative on only 1% of the TrackML dataset which is far too small to draw any conclusions.

Many more research projects exist based on the TrackML dataset and this demonstrates the successful challenge format. A key aspect of the popularity of the TrackML dataset was and is the simplicity of the dataset format that enabled fast prototyping. This format along with the detector layout are discussed in the next section.

6.3 The TrackML Challenge

The innovative concept of data science competitions started mostly with the Kaggle platform [11] in 2009. A data competition platform allows its users to take part in machine learning competitions, share notebooks and discuss solutions in forums. The competitions are generally proposed by large companies to solve a specific problem they are facing and for that, they publish their data and a problem description. These companies *challenge* the largest public data community offering significant money prizes. Kaggle for example is the largest and most diverse machine learning community in the world comprising the world's most renowned experts.

TrackML is the third HEP challenge on the Kaggle platform with the Université de Genève as a platinum sponsor of the event. The earlier challenges (whose organizing committee overlaps with the TrackML's) are the Higgs Boson challenge [12] and the Flavor of Physics challenge [13]. The former aimed at recognizing novel signal events from common background events and the latter challenged participants to compensate for systematic bias introduced by simulators of events. Contrary to previous ML-HEP challenges and to most ML challenges, the TrackML was organized in two different phases: An accuracy phase on Kaggle and a throughput phase on Codalab [14].

[2]The winner had a score of 94.4% in 0.56 second but will not be covered in this thesis as the approach is an optimized combinatorics algorithm

The TrackML dataset formed the starting point of the approach proposed in this thesis. The same data structure and properties described in the following sections are used to demonstrate ML based approaches in Chapter 7.

6.3.1 TrackML Detector Layout

The detector model is built to simulate measured hits similar to what is expected for HL-LHC. It is inspired from both the ATLAS and CMS upgrade tracker designs. The coordinate system is similar to the one presented in Chapter 2.2.1. The detector geometry with its three sub-detectors is presented in Figure 6.3. The inner-most sub-detector is the pixel detector with a spatial resolution of $50\mu\text{m} \times 50\mu\text{m}$. Short Strips (red in Figure 6.3) have a resolution of $80\mu\text{m} \times 1200\mu\text{m}$ while the long strips (green) have a resolution of 0.12mm $\times$ 10.8mm. The pseudo-rapidity range covered is $|\eta| < 3$.

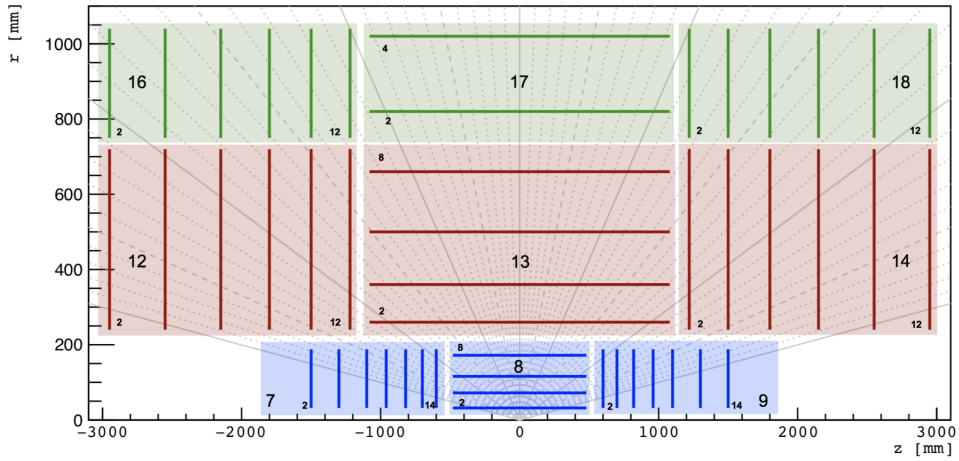
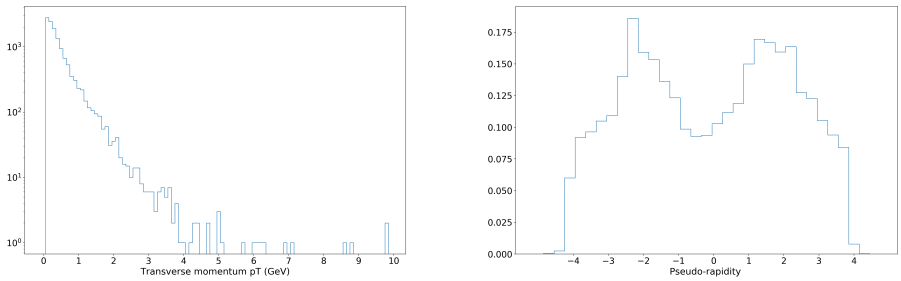


Figure 6.3: Layout of the TrackML detector with colors showing the three sub-detectors (pixels, short and long strips) and numbers indexing volumes (large font) and layers (smaller font per volume).

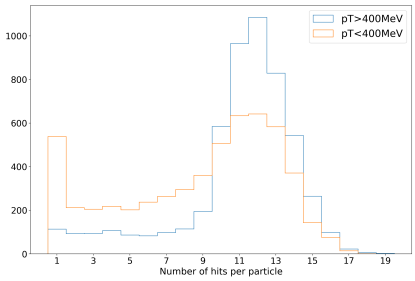
The dataset simulates a $t\bar{t}$ pair production (similarly to the ITk event used in Chapter 8) and overlays a pileup of $\mu = 200$. The collisions are generated using Pythia 8[3] [15] and the particles are propagated using ACTS[4] fast simulation [16]. The particles simulated are embedded in an inhomogenous magnetic field, i.e. particles bend differently depending on the detector region they are in. The magnetic field was not provided to participants as an additional challenge to the robustness of proposed models. A minimum transverse momentum cut of 150MeV was applied which is lower than the standard ATLAS cut of 400MeV. Noise hits, i.e. hits without particle label are also included along with multiple scattering (distorted tracks) and inactive sensors (holes). The resulting dataset contains roughly 100K hits associated to 10K particles per event. The transverse momentum, pseudo-rapidity and particle size distributions are shown in Figure 6.4. These three distributions were selected as they represent the quantities having the highest impact on the proposed approach of similarity search. A more complete track parameters overview of the TrackML challenge can be found in [2].

[3]Standard software for the generation of events in high-energy collisions

[4]ACTS encapsulates track reconstruction software into a generic and framework (experiment) independent package



(a) (b)



(c)

Figure 6.4: Relevant track parameters per TrackML event. 6.4(a) shows the transvere momentum distribution in log scale. The majority of particles (pileup) have $P_T < 1$GeV. 6.4(b) shows the density distribution along the pseudo-rapidity η and 6.4(c) the number of hits per particle. Two distributions are shown to highlight the difference between low P_T and particles (< 400MeV) and high P_T ones. The first bin represents the noise hits added per event. On average a particle has 11 hits in the TrackML dataset (> 400MeV).

6.3.2 Data Files and Setup

The dataset format adopted for the TrackML challenge is the successful encoding of nested and complex information into a friendly, machine learning adapted file format. Generally specialized tools and parsers are needed to process particle physics events. The csv (comma separated values) format of the challenge is on the other hand easy to digest by any tool or even read as plain text. Most machine learning competitions on Kaggle use this format which enabled participants to process the data without mastery of physics software. Each event consisted of four csv files :

1. **Hits** : A List of data points (hits) represented by their id, x, y, z coordinates and volume/layer indices.

2. **Cells** : A List of activated pixels forming clusters. Each line is mapped to the corresponding entry in the hits file and contains a variable list of pixel coordinates (two dimensional) and the deposited charge. A hit can have one or multiple cells associated with it.

3. **Particles** : A list of the particles produced. Each entry has an id and parameters information (momentum and vertex)

4. **Truth** : A list mapping the hit id to its particle id (labels) with corresponding track parameters and a symbolic weight for the scoring (discussed in the next section).

Participants to the challenge had access to 100K of such events to create a model that associates hits to particles. Once the model is built, it runs on a specific set of 125 files (refered to as test files) that contain only hits and cells information. The model outputs a list where each entry contains the hit id and an arbitrary generated track id. For technical reasons all test events had to be merged in a single file comprising an event id, a hit id and the associated track id. Participants then created a submission and loaded their solutions on the Kaggle platform. Internally, Kaggle ranked solutions using the hit-particle associations that it stores (concretely particle and truth files of the test events).

In the first phase of the challenge, each submission is *scored* according to its quality.

6.3.3 Scoring Solutions

As mentioned throughout this thesis, tracking is a clustering problem. Hits receive labels that regroup them. A perfect algorithm groups hits according to their true particle id and a straightforward performance measure is the comparison of the algorithm's labels with the actual particle id. Metrics such as the Adjusted Rand Index (ARI) perform exactly this comparison. The ARI is a function that measures the similarity of the two assignments, ignoring permutations. A perfect hit-track association has a score of 1.

Although this scoring succeeds in ranking perfect solutions, what about solutions that get a score of 0.9 or 0.7 ? Can we interpret the physics quality of such tracking models? Is a 0.9 solution missing 10% of particles or rather having all tracks with only 90% purity? These, in physics performance, are wildly different solutions[5]. The TrackML scoring function was designed to incorporate such physics aspects.

Each hit is weighted by its *contribution* to the track. Figure 6.5 illustrates an example track with the different weights categories of its hits. The reasoning behind using weights is straightforward. A model is penalized less if it tends to loose hits with low weights because their loss does not compromise the track parameter estimates which are at the end, the very purpose of tracking. For example, if one removes the fourth and/or fifth hits starting from the inner most layer, the particle trajectory does not change much when connecting the remaining hits. On the other hand, loosing the outermost hit heavily affects the momentum resolution whereas loosing the innermost hit compromises the vertex finding. The weights are also adjusted with respect to the particle size.

The scoring procedure is detailed in Algorithm 1. The list of good tracks is produced per event and used in the global scoring as described by equation 6.1.

$$S = \frac{1}{N_{events}} \sum_{events} \sum_{good\ tracks} \sum_{i=0}^{M} w_i \tag{6.1}$$

where w_i is the weight (summing up to 1) of the ith hit in the track and M the size of the track. If the found track matches perfectly a true particle, the track score is 1. The final score is normalized between 0 and 1. A proposed model can leave a number of hits without track assignment (noise hits) and these do not contribute to the score.

[5]Charged particle tracking prefers the second scenario as track parameters can easily be retrieved with 90% of a particle content but loosing 10% of all particles is unacceptable

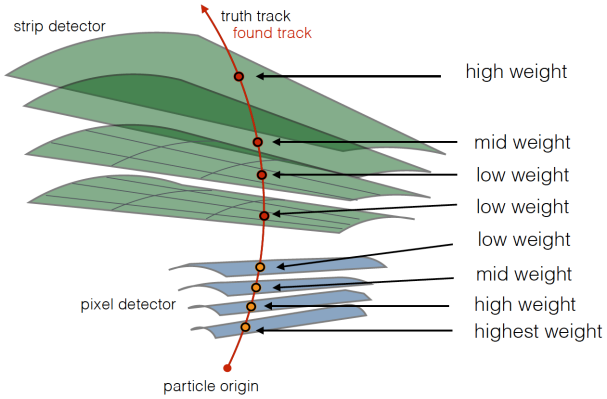


Figure 6.5: Ranking of the hits depending on their order. Higher weights translate to higher penalties if the corresponding hits are lost.

Algorithm 1 TrackML scoring function

Input : List of tracks T
Output: List of good tracks $GoodTracks$
  for t in $Tracks$ **do**
    track=MajorityLabel(t) //Get the leading particle in the track
    M=size(track)
    T=size(t)
    if $M \geq 4$ **then**
      particle=GetParticle(track) //Get the associated true particle
      P= size(particle)
      if $P \geq 4$ **then**
        ParticlePurity=$\frac{M}{P}$

        TrackPurity=$\frac{M}{T}$

        if $TrackPurity \geq 0.5$ and $ParticlePurity \geq 0.5$ **then**
          $GoodTracks \leftarrow t$
        end if
      end if
    end if
  end for

This scoring function was kept for the second phase of the TrackML challenge that targeted reconstruction speed. Both the solution quality score described above and the model speed formed the two dimensional scoring metric used in the second phase.

6.3.4 Competition Results

The accuracy phase run from 30 April to 13 August 2018 on Kaggle. Within this challenge, the speed of the approach did not matter in the scoring or in the final results (Kaggle leaderboard). The three best solutions received the money prize and three additional approaches were picked by the jury for their innovation and creativity. The winners of the first phase with scores around 90% are :

- "Top Quarks", an industrial mathematics student who proposed a rather conventional tracking algorithm consisting of : Pair seeding, triplet extension, trajectory following, track cleaning. He used machine learning

classifiers to discard bad quality combinations.

- "Outrunner", a software engineer in deep learning. He built a deep neural network to predict every and any possible hit pair connection probability, i.e. full event adjacency matrix.

- "Sergey Gorbunov", a physicist and expert in tracking. He used a traditional tracking algorithm consisting of triplet seeding and trajectory following. He is the winner of phase 2 and therefore of the full TrackML challenge.

The jury prizes were awarded to approaches relying on hough transform, DBSCAN clustering and LSTM deep learning.

In the second phase of the challenge, the two first winning approaches were both relying on traditional tracking methods optimized for speed with C++. The approach that won the third place was a Directed Acyclic Graph (DAG) resulting in a 92% accuracy but with a time execution of 7.2 seconds. The winning approach takes only 0.56 seconds per event.

A complete study of the proposed approaches in both phases is available in [2]. Listed below are the conclusions agreed upon in the community:

- A good score translated to good physics performances which demonstrates the validity of the metric used especially that the only alternative is to look manually at many distributions.

- The winning solutions that were physics inspired had to optimize a large (believed to be thousands) number of parameters by hand. This signals the urgent need for automatic parameter estimation techniques.

- The second phase had 10 active participants on average against hundreds in phase one. This highlights the impact of the platform popularity but also that the machine learning community is not necessary interested or specialized in performance coding. This might change soon as Kaggle is working on enabling software submissions in order to evaluate the speed.

Thoughts on the lack of ML solutions

The fact that the winning solution is not a machine learning based one does not mean that tracking cannot be solved with machine learning or that machine learning is not mature enough. Moreover, the fact that a rather intuitive combinatorial approach was the winning solution only means that tracking is an intuitive problem that can be solved by an iterative hand tuned algorithm.

Tracking is not a hard problem but rather a multi-step one. An ML model might have better success if it is evaluated on each one of these steps rather than the full chain. For example, the TrackML challenge rewarded full particle finding (all good particle) although the seeding step could be considered an important first achievement. It has been demonstrated that a simple triplet classifier can produce less fakes (less combinatorics) compared to a traditional cut based approach [17][18]. If the score rather builds on multiple stages, we could potentially see that machine learning can already perform better in some parts (seeding, filtering) rather than a full chain.

This thesis is about the use of a machine learning model(s) for charged particle tracking.

Bibliography

[1] Amrouche, Sabrina, et al. The Tracking Machine Learning Challenge: Accuracy Phase [Book Chapter]. No. arXiv: 1904.06778 v2. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2019.

[2] Amrouche, Sabrina, et al. "The tracking machine learning challenge: Accuracy phase." The NeurIPS'18 Competition. Springer, Cham, 2020. 231-264.

[3] Peterson, Carsten. "Track finding with neural networks." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 279.3 (1989): 537-545.

[4] Abt, I., et al. "Cellular automaton and Kalman filter based track search in the HERA-B pattern tracker." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 490.3 (2002): 546-558.

[5] Connecting the Dots 2019 edition `https://indico.cern.ch/event/742793/`

[6] Tüysüz, Cenk, et al. "A Quantum Graph Neural Network Approach to Particle Track Reconstruction." arXiv preprint arXiv:2007.06868 (2020).

[7] The HEP.Trk Collaboration `https://heptrkx.github.io/`

[8] The Exa.TrkX Collaboration. HEP advanced tracking algorithms at the exascale`https://exatrkx.github.io/`

[9] Farrell, Steven, et al. "Novel deep learning methods for track reconstruction." arXiv preprint arXiv:1810.06111 (2018).

[10] Tüysüz, Cenk, et al. "Particle Track Reconstruction with Quantum Algorithms." arXiv preprint arXiv:2003.08126 (2020).

[11] The Kaggle plateform `https://www.kaggle.com/`

[12] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Keegl, andD. Rousseau, "The Higgs boson machine learning challenge," in NIPS 2014 Workshop on High-energy Physics and Machine Learning. `http://proceedings.mlr.press/v42/cowa14.html`

[13] Flavours of Physics: Finding $\tau- > \mu\mu\mu$ `https://www.kaggle.com/c/flavours-of-physics`

[14] Accelerating reproducible computational research `https://codalab.org/`

[15] Sjöstrand, T., Mrenna, S., Skands, P.: A brief introduction to PYTHIA8.1. Computer Physics Communications178(11), 852–867 (2008).DOI 10.1016/j.cpc.2008.01.036

[16] Gumpert, C., Salzburger, A., Kiehn, M., Hrdinka, J., Calace, N.: ACTS:from ATLAS software towards a common track reconstruction software.J. Phys. Conf. Ser.898(4), 042011 (2017). DOI 10.1088/1742-6596/898/4/042011

[17] Amrouche, Sabrina, et al. "Hashing and metric learning for charged particle tracking."

[18] Dietrich, Felix. "Track Seed Classification with Deep Neural Networks." arXiv preprint arXiv:1910.06779 (2019)

7

Similarity search for charged particle tracking

Fast search techniques allow to rapidly access a group of data points that share common properties. Similarity search, uses supervised knowledge to improve the meaning of the properties shared and align it with the definition of charged particle tracking. Similarity search therefore allows a fast access to a combination of points that likely form a particle. Different combinations of algorithms are evaluated both for fast search (Approximate Nearest Neighbors) and similarity patterns (deep learning).

Similarity search is first proposed and evaluated on the TrackML challenge dataset. A second version relying on a novel metric learning model is later described for the ITk dataset (Chapter 9). In this chapter, we describe the application of ANN techniques for tracking. Metric learning is then applied to improve the content of the obtained buckets of neighbors, allowing to retrieve, often, complete traces. Relevant results have been peer reviewed and published in IEEE 2019 [1] and NeurIPS 2019 [2].

7.1 Definitions and Notations

When traversing the detector, every p_i particle generates a set of hits h_j:

$$p_i = \{h_0, .., h_j, .., h_{n_i}\} \quad where \quad h_j \in \mathbb{R}^d \tag{7.1}$$

The ensemble of particles generated in a collision is $P = \{p_i\}$. Each hit is a d dimensional feature vector that encodes the detector readout. Some tracking applications rely only on the geometrical global coordinates of a hit, thus $d = 3$. Depending on the use case and the model used, this number will vary. If the cells generated by a particle are used as features, each hit will be additionally represented by a variable size 3D image. The image encodes the cells shape information and the charge deposited per sensor.

A tracking model returns the ensemble T, a list of m sets of hits:

$$T = \{t_0, ...t_m\} = \{\{h_{00}, .., h_{0n'_0}\}, .., \{h_{m0}, .., h_{mn'_m}\}\} \tag{7.2}$$

where n'_i is the size of a track t_i. The tracking algorithm goal is to return a set T such that :

$$T = P \tag{7.3}$$

Equation 7.3 can never be achieved in practice and various performance metrics (introduced in Chapter 3) are defined to quantify the closeness of T and P. In a TrackML event, $|P| \approx 10000$ with ≈ 100000 hits.

An ANN query q in an event returns a list of B hit candidates referred to as a *bucket*.

$$ANN(q) = \{h_0, .., h_B\} \text{ associated to particles } \{p_i, .., p_j\} \tag{7.4}$$

The leading particle size in a bucket is defined as the most common particle label associated to the hits. For example, a bucket of $B = 10$ hits with associated particle labels $\{p_{10}, p_{10}, p_{10}, p_{10}, p_{10}, p_{10}, p_{10}, p_5, p_2, p_2\}$ has a leading particle size of 7 hits, i.e. p_{10} is the leading particle. The distribution of leading particle sizes computed from a large number of ANN queries is an important metric for evaluating buckets quality. If a bucket contains less than 4 hits from the same particle, it is considered as noise. Such noise buckets should be minimized in order to improve tracking performances.

In order to evaluate the performances of the proposed approach, the relevant metrics are summarized in Table 7.1.

Quantity	Purity	Efficiency	Event Efficiency	Leading Particle Size	Noise	Seed
Definition	Majority of hits sharing the same particle identifier over total hits	Majority of hits sharing the same particle identifier over true particle total hits	Fraction of particles with efficiency >80% in an event	Size of the highest purity track in a bucket	Total hits matched to a particle by less than 4 hits	>3 hits define a seed. A particle is reconstructable

Table 7.1: Summary of parameters and metrics definition.

7.2 Proposed Approach

We propose to combine ANNs with metric learning to rapidly lookup hits produced by the same particle. A TrackML event is projected into a new feature space where tracks are separated and well clustered. The ANN index is built on this new feature space to maximize the probability of containing a particle in every queried bucket.

The proposed approach is summarized in algorithm 2. The function $metricLearn()$ is a supervised metric learning model that maps a set of hits to a new feature space where particles are easily found. $buildIndex()$ is the implementation of an ANN index from a collection of data points with a given metric. In Algorithm 2, the ANN index is built from *mapped* hits using an euclidean distance. The index can also be built on the raw features of the hits (such as global coordinates). In this case, the angular distance results in better results than the euclidean distance. Once the hits are mapped and stored in an index, we can either randomly query buckets (as in Algorithm 2) or query all the possible hits in the index. Various trials have shown that random queries on a subset of the index achieve similar performances as compared to querying the full index. This only applies to the TrackML dataset as demonstrated in the following chapter.

The function $cluster2Tracks()$ is an agglomerative clustering that merges close-by points until reaching a pre-defined maximum distance.

Classical tracking relies on a function similar to $cluster2Tracks$[1] in large manually defined bins of up to 800 hits. The main idea behind using ANNs is to run this CPU intensive function on small event-defined buckets. Running a function on many small chunks of the data is more effective than running it on larger ones or on the full event. An illustration of this argument is

[1] Clustering and combinatorial approaches both compute a large number of distances (roads in the case of combinatorics) before converging.

Algorithm 2 Tracking with ANNs and metric learning

Input : List of hits H
Output: List of tracks $foundTracks$
Require: $T \leftarrow cluster2Tracks(H)$
  $P \leftarrow metricLearn(H)$
  $ANN \leftarrow buildIndex(P, metric)$
  $foundTracks \leftarrow \emptyset$
  while $i \leq Nqueries$ **do**
    $n \leftarrow random()$
    $bucket \leftarrow ANN.query(n)$
    $t \leftarrow cluster2Tracks(bucket)$
    $foundTracks \xleftarrow{+} t$
    $i \leftarrow i + 1$
  end while

shown in Figure 7.1. Such clustering emulates well the exponential behavior of combinatorics when the dataset (or bin) size increases.

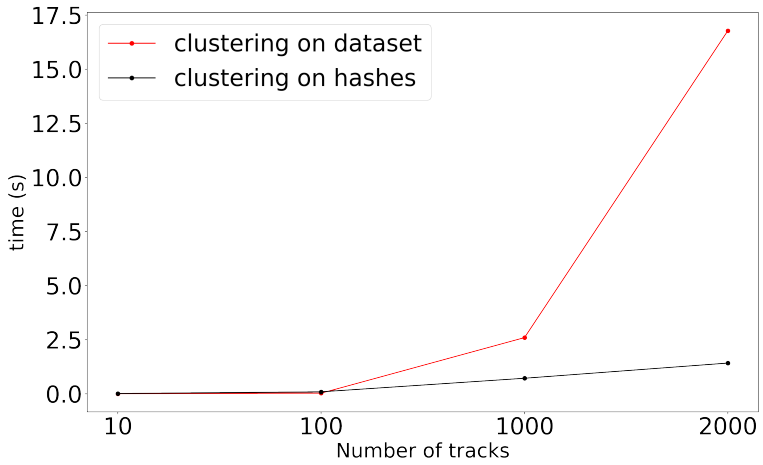


Figure 7.1: Scaling of an agglomerative clustering run on a full dataset compared to the same function run on ANN buckets.

The dataset considered contain 10, 100, 1000 and 2000 unique particles. Larger datasets could not be considered as the agglomerative clustering large distance matrix could not fit in memory. The buckets have fixed sizes of 50 hits and the total number of queries performed per dataset is $N_{queries} = \frac{1}{5}N$ where N is the size of the dataset. This number was found to be the optimal ratio that produces both minimal number of buckets while containing all the particles in the dataset.

Running the clustering on ANN buckets built from 10K particles takes approximately **2.8** seconds on a single thread[2]. If 10 threads are used instead, the bucketing and clustering is expected to run 10 times faster. On GPUs, it is possible to run a thread per bucket resulting in thousands parallel queries and clustering each taking approximately 2ms. An equivalent study that uses the ATLAS track finder instead of a clustering algorithm, is conducted on Chapter 8.

[2]Tests performed on a personal computer,Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, using python.

In the remainder of this chapter, the ANN index and the metric learning function are detailed with relevant results from the application on the TrackML dataset.

7.2.1 Indexing Charged Particle Hits

Different ANN techniques are described in Chapter 5. In this section, a tree based and a graph based ANN structures are used to index hit in an event. The metric used to build an ANN index from raw data (hits positions) is the angular distance while the euclidean distance is used after mapping the dataset with metric learning. Both the tree and graph structure provide similar buckets (hits content) but show different query time performances.

An example of an ANN bucket is shown along the transverse plane in 7.2. To obtain this bucket, a TrackML event is indexed with an angular distance applied to the hits global positions (x,y,z). A query point is (randomly) selected within the hits of the event while specifying the number of neighbors to return.

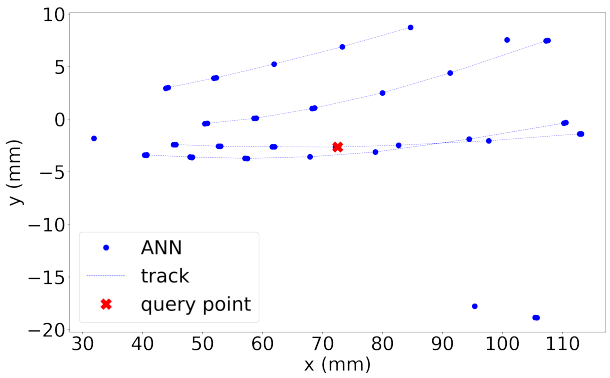


Figure 7.2: Example of an ANN bucket using an angular distance on global hits coordinates.

In Figure 7.2, the bucket size is 50. The true tracks contained in the bucket are represented by dotted lines. Because the bucket contains multiple tracks with sizes well above four hits, it can be considered a good bucket.

Figure 7.3 shows the distribution of the leading particle size per bucket for a bucket size of 20 and 50 hits using the angular distance. A bucket is constructed for every hit in the event. The unique particle sizes distribution is also shown in the Figure. This quantity describes the maximum size found for each particle in the event. The highlighted leading particle size distributions do not accurately describe the content of a bucket but they rapidly inform us whether a bucket contains a reconstructable track or not. Indeed, buckets with a leading particle size of less than 3 hits cannot be reconstructed as they cannot form a seed. The figure also shows the impact of the bucket size. Larger buckets increase the probability of containing more hits from the same particle. We can see from Figure 7.3 that already an ANN with an angular distance and a bucket size of 50 is able to index seeds (3 hits), tracklets and even larger (complete) tracks.

In the next section multiple ANN libraries are evaluated on the TrackML dataset for speed and bucket quality.

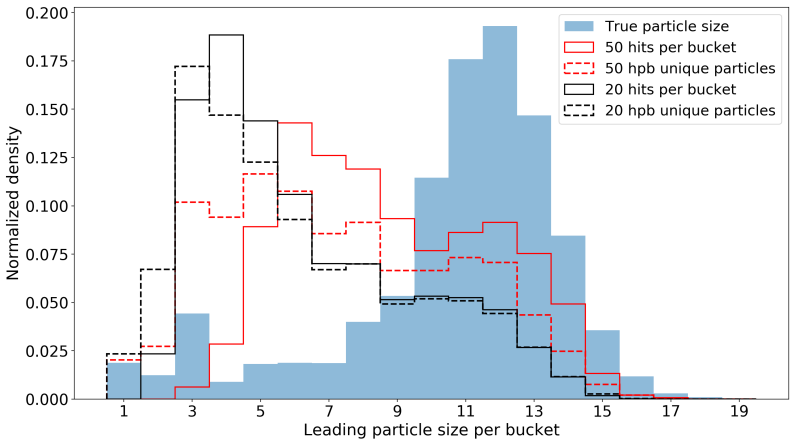


Figure 7.3: Distribution of the leading particle size per bucket for different bucket sizes. The distribution of actual particle sizes per event is highlighted in the background of the figure and the unique leading particle size distribution for the two bucket sizes is shown as discontinuous lines.

7.2.2 ANN Techniques Evaluation

We introduce a quality measure that combines two essential quantities in an ANN bucket:

- Number and size of reconstructable particles.
- The proportion of noise hits per bucket.

For a query point q, the previous quantities are encoded in the following metric :

$$Quality(q) = \frac{\sum S_{RecoParticles}}{N} - p_{noise} \tag{7.5}$$

where $S_{RecoParticles}$ represents the size of reconstructable particles in each bucket, N is the number of reconstructable particles and p_{noise} the percentage of noise hits per bucket. When p_{noise} is small, the quality tends to describe the average particle size per bucket (which has to be maximized). In Figure 7.2, a total of 5 hits do not contribute to any particle which results in $p_{noise} = \frac{5}{50}$. The number of reconstructable particles $N = 4$ and $\sum S_{RecoParticles} = 45$. The quality metric for this bucket is therefore ≈ 11 which is very close the average track size we see in Figure 7.2. The second evaluation metric is the speed per query or throughput, i.e. number of queries per second. As seen in Chapter 5, ANN algorithms are designed to maximize throughput. For evaluation, we use a full TrackML event considering the three dimensional features (x, y, z). The ANN index is built using an angular distance and the queries are selected randomly. Different bucket sizes are tested to highlight the impact on performances.

In Figure 7.4, three different ANN algorithms are compared. Description of the techniques is presented in Chapter 5. Annoy is a tree based technique, Hnsw a graph based algorithm and Faiss relies on quantization and inverted file index. The two performances axes are shown in the figure : average speed per 10K ANN queries in milliseconds (x axis) and the quality measure introduced in Equation 7.5 (y axis) also average over 10K queries.

Different techniques are color coded and the marker shape denotes different bucket sizes. On average Hnsw is the fastest technique (significantly faster

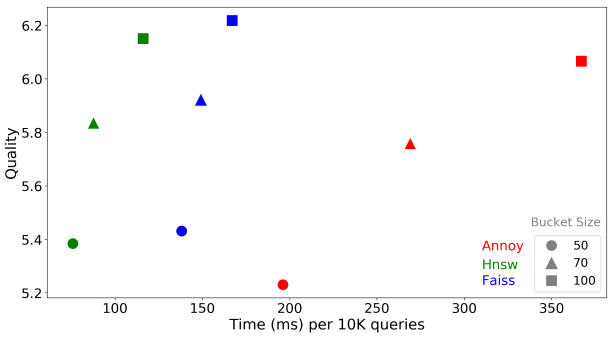


Figure 7.4: Comparison of three ANN techniques along the bucket quality and time per 10K queries dimensions. Different bucket sizes are used to illustrate the time and quality impact.

than Annoy). The quality of the buckets is slightly higher for Faiss however. Using an angular distance, the three techniques return approximately similar buckets except for one or two hits difference at most. A detailed study of the Hnsw buckets and their correlation to charged particles can be found in [6].

7.2.3 ANNs Performances on CPUs and GPUs

Most ANN techniques are designed to allow running multiple queries at once. This behavior is referred to as batch mode and is often faster than running queries one after the other. Moreover, batch mode is suitable for GPUs that naturally parallelize the search in the index. Since the buckets are independent, i.e. finding a particle in a bucket will not depend on other buckets, it is possible to heavily parallelize the inner loop of Algorithm 2. When considering parallel bucketing, a duplicate removal strategy is necessary when all the outputs of the queries are returned. A discussion on duplicate rate and duplicate removal is presented in Chapter 8. Figure 7.5(a) shows the different scaling of ANN queries on GPU and on CPU. The buckets on CPU are built in a for loop, one after the other to emulate a single thread (process). On GPUs however, the buckets are produced in parallel through the batch mode which brings down the total bucketing time by a factor of 56. Figure 7.5(b) and Figure 7.5(c) show the total time as a function of the number of queries and the potential efficiency using truth tracking. The efficiency is the fraction of contained particles in the queried buckets by the total number of particles in an event. An efficiency of 95% is reached after 4K ANN queries which takes around 3 milliseconds on GPU. Additionally, the clustering cost per bucket must be added to this timing in order to get the approximate tracking time.

7.2.4 Learning a Tracking Representation

We propose the use of supervised metric learning techniques to map the event data into a new feature space where hits that belong to the same particle are forced into close-by coordinates. This section contributes the ***similarity*** of the chapter's title. Metric learning is at the core of the proposed approach within this thesis. It is the portion of the solution that effectively uses simulation labels to extract a particle pattern and generalize to unseen collision events. In this section, we focus on the application of Local Fisher Discriminant Analysis (LFDA) and Triplet Neural Nets to the TrackML dataset.

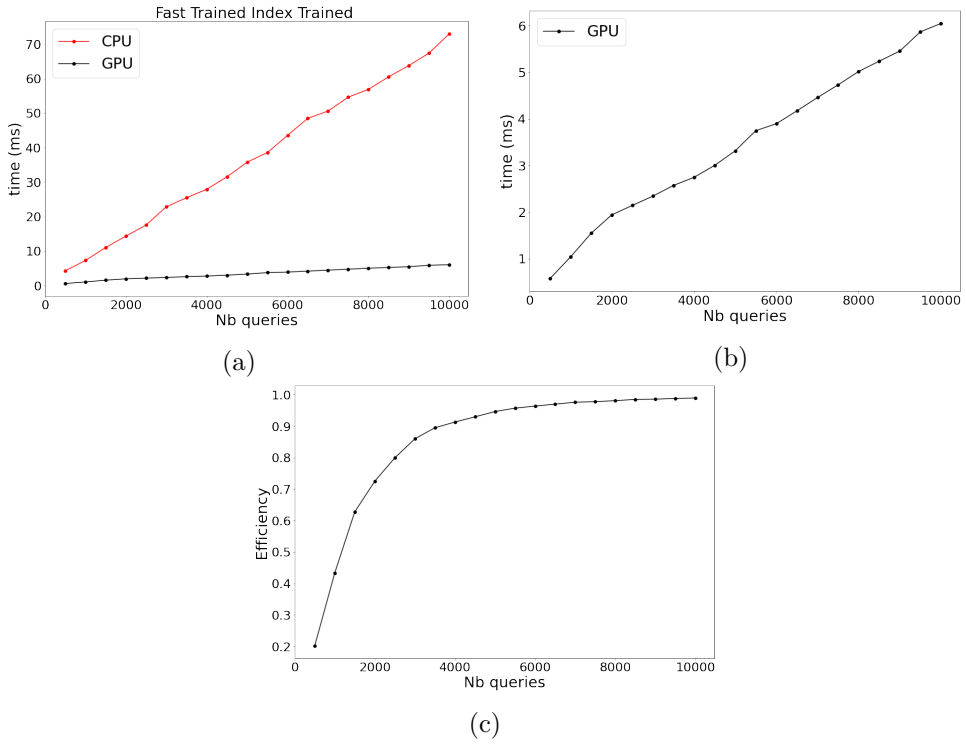


Figure 7.5: Scaling of the query time on GPU and CPU. In (a) the scaling of ANN queries on GPU is compared to the CPU alternative. (b) shows the GPU scaling only to emphasise the time difference. (c) shows the efficiency as a function of the GPU queries.

LFDA for particle representation learning

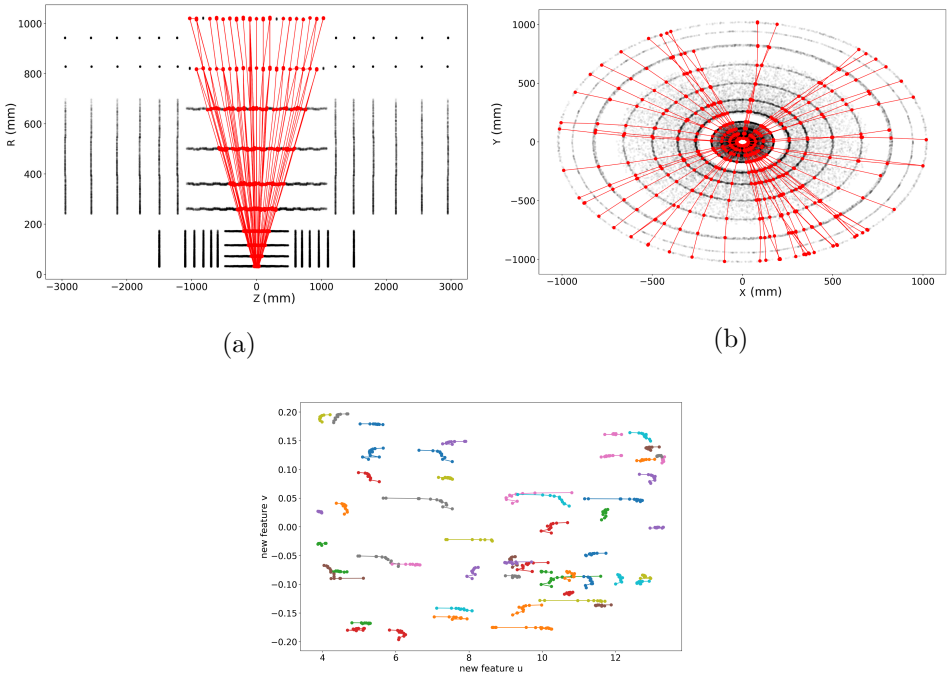
We use LFDA to learn a mapping from raw event data to a two dimensional *simplified* space where the application of a clustering technique can easily retrieve tracks. LFDA performs linear mappings and the best results were obtained with the raw features : $\sqrt{x^2 + y^2}$ and the inner angles θ and ϕ. The LFDA model is validated on unseen events. The resulting mapping of a subset from a validation event is shown in Figure 7.6.

This subset (shown in red in Figures 7.6(a) and 7.6(b)) includes all the hits with $|\eta| < 1$ and particles $P_T > 2$ GeV. Figure 7.6(a) shows the longitudinal view of the selected subset with the remaining hits (black dots) while Figure 7.6(b) shows the transverse view of the same hits. The LFDA mapping to a 2D space is illustrated in Figure 7.6(c). The actual (true) particles in this mapping are denoted by continuous lines of different colors. For visibility purposes, only tracks with at least 4 hits are shown.

The separation between tracks is easily noticeable and using a clustering algorithm on top of that mapping allows one to retrieve the full particles in this subset.

Clustering representations

We propose to run a clustering algorithm on the learned projections (u,v) illustrated in Figure 7.6(c). Both KMeans and Agglomerative clustering are studied and tested in this use case since the particles are well separated. KMeans requires the number of clusters (particles) to be known in advance while Agglomerative clustering allows to specify a threshold above which



(a) (b)

(c)

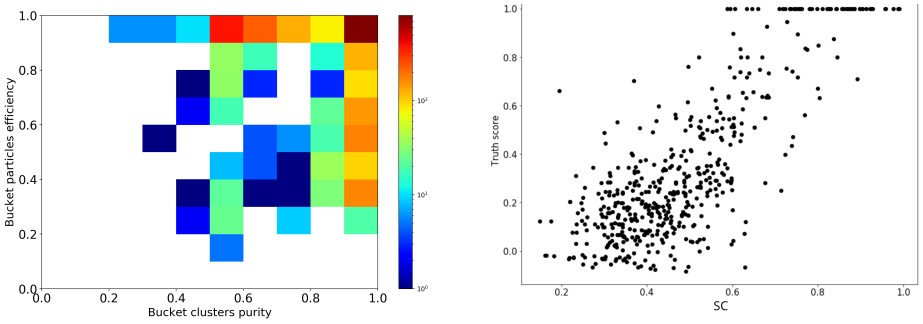
Figure 7.6: Application of LFDA metric learning on a subset of TrackML event. The longitudinal projection of the subset is shown in (a) while the transverse view is shown in (b). The 2 dimensional output space is shown in (c) where same particle hits have the same color and are linked by continuous lines.

points (hits) are not merged into new clusters[3]. Unless specified otherwise, the clustering algorithm is a KMeans with default parameters and a cluster number of 3. This choice is motivated by the bucket size and the average particle size in order to maximize the size of found particles. The clustering algorithm is run per bucket and two quantities are measured :

- **Cluster purity** : Ratio of the majority particle size by the cluster size.
- **Particle efficiency** : Ratio of the majority particle size by the total true particle size.

The purity describes whether a cluster contains a unique particle (=1) or multiple (>1). The efficiency is similar to the classical HEP definition. An efficiency of 1 means that the cluster contains the full particle. Figure 7.7(a) shows the 2D distribution of purity and efficiency for all buckets of 20 hits, with a P_T cut of 1GeV and a minimum track size of 4 hits. As can be seen from the 2D histogram, the majority of 20 hits buckets contain perfectly separated particles with purity =1 and efficiency =1, i.e. the clusters extracted with KMeans contain the full trace of particles. Second most common buckets have perfect efficiency and a purity $> 60\%$. These are clusters that contain outlier hits from additional particles while containing a full (short) particle. Buckets with 100% purity and an efficiency $> 40\%$ show opposite behavior. Despite the clusters containing a single particle, the cluster size is smaller than the actual particle size. This is also caused by the bucket inefficiency that do not necessarily contain the full particle.

[3]This is useful when the clusters exhibit a non flat geometry,i.e. different shapes.



(a) (b)

Figure 7.7: Cluster quality distributions. The joint efficiency and purity of the clusters is shown in Figure 7.7(a). The silhouette coefficient correlated with the true clustering score is shown in Figure 7.7(b)

.

Some buckets contain clusters with low purity and low efficiency. Conveniently such buckets have different geometrical shapes compared to the ones containing full particles, i.e. the clusters formed within each bucket are not well separated. In order to detect these cases, we define a shape metric at the bucket level referred to as the homogeneity value. The homogeneity is a metric that describes how well defined are the clusters. A cluster is well defined if the distances amongst its elements are much smaller than the distance between its elements and other clusters. We commonly refer to the average distances within a cluster as compactness CM (the smaller the distances, the more compact the clusters) and the average distances between different clusters as isolation IS. Thus, well defined clusters are compact and isolated. The silhouette coefficient (SC) combines these metrics in the following formula:

$$SC = \frac{IS - CM}{max(IS, CM)} \tag{7.6}$$

This quantity is computed for every sample and the final score for a given clustering output is the average over all samples. Figure 7.7(b) shows the correlation between a bucket clustering true score and the silhouette coefficient metric. We use the truth information to compute the true score of a clustered bucket. The correlation shown in the figure suggests that compact clusters contain more particles. The SC is then used to discard buckets with poorly defined clusters of multiple particles or incomplete traces. For this study, a cut of SC=0.8 is chosen.

The complete tracking strategy is then summarized in the following steps:

1. The raw event is mapped to a new two dimmensional feature space with LFDA.

2. Buckets of 20 (or 50) hits are built from the resulting mapping.

3. KMeans is applied on every bucket.

4. Clusters from buckets with a $SC \geq 0.8$ are readout as reconstructed tracks.

To compute the event efficiency, the reconstructed tracks are compared to the true particles given an acceptance threshold (usually 80%). The ratio of correctly reconstructed tracks to the full event tracks is the final event efficiency. Figure 7.8 shows the different efficiency ratios for acceptance threshold

between 50% and 100%. The different colors indicate different bucket sizes and the different markers designate the presence or absence of a P_T cut at 500 MeV. A total of 100K buckets have been queried.

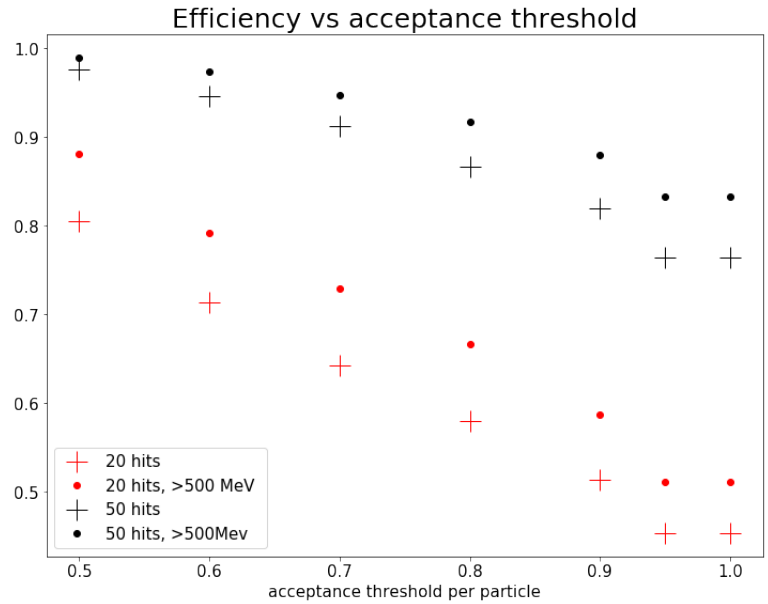


Figure 7.8: Event efficiency as a function of the particle purity acceptance threshold for 100K buckets.

Given the standard purity threshold of 80%, the best efficiency is given by buckets of 50 hits with 88% on the full event and 94% with a P_T cut of 500 MeV. We show different purity cuts to highlight the potential of the approach for seeding. Nearly all the particles are retrieved when considering a threshold of 50%. This is on average 5 hits per particle which, if used as initial seeds, would alleviate the combinatorial seeding time.

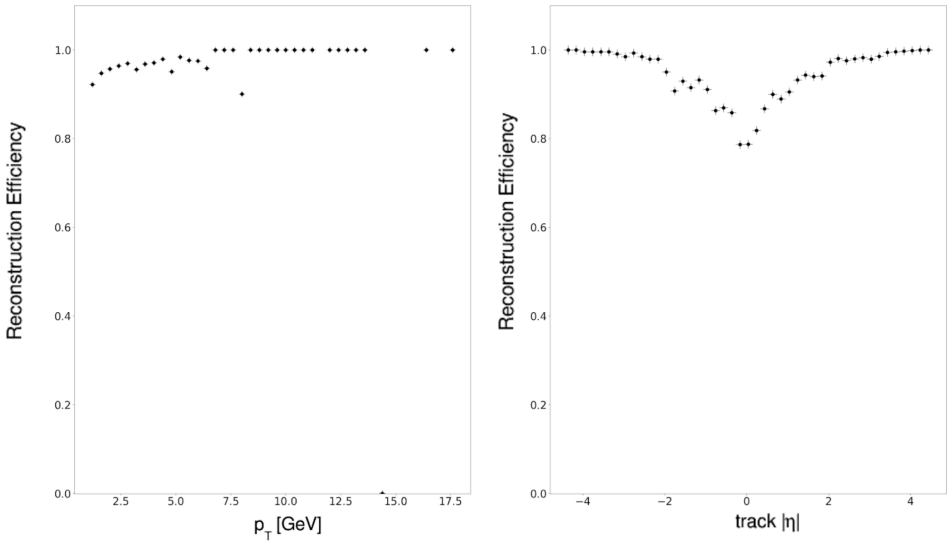


Figure 7.9: Efficiency of the approach as a function of P_T (GeV) and pseudo-rapidity (η).

Figure 7.9 shows the efficiencies as a function of P_T and eta averaged over 30 events with an acceptance cut of 80%. The small variations in the efficiency against P_T can be attributed to a low number of statistics in the

high P_T regime. In the central pseudorapidity region ($|\eta| < 2$) however, the inefficiencies are due to the nature of the tracks that are not properly mapped by the metric learning algorithm. This can be explained by the layout of the detector that results in simpler tracks in the forward region. As will be discussed in future chapters, this not a feature of the TrackML detector only but also extends to the ATLAS Inner Tracker Phase II (discussed in the next chapter).

Generally this behavior of model performance deterioration in the central detector region is consistent independently of the specific ML algorithm used. In Chapter 9, it is solved by training different models on the different pseudorapidity regions.

7.3 Summary and Conclusions

Similarities between a commercial search engine and charged particle tracking in an LHC experiment might not be obvious. Seen from afar however, both applications have the same crucial need : retrieve data points that share similar properties as fast as possible. In this Chapter, we explored the definition and evaluation of Approximate Nearest Neighbors for charged particle tracking on the TrackML dataset. The obtained buckets of hits, defined using the angular distance, contained at least one reconstructable track. Additionally, it was shown that the size of the contained track depends on the number of considered neighbors. The evaluation of buckets comprised CPU vs GPU tests as well as different types of indexing techniques : Tree structure (Annoy), graph structure (Hnsw) and inverted file indexing (Faiss). Hnsw yielded the fastest results closely followed by Faiss. The proposed quality metric included the average leading particle size per bucket as well as the fraction of noise hits. This metric slightly favored the Faiss search algorithm. Moreover, the Faiss library includes a GPU implementation that allowed to emulate a fully multi-threaded bucketing of hits: 4K queries in approximately 3ms with a 95% efficiency.

Learning the similarity metric not only allowed to improve the quality of the buckets but also enabled the use of a clustering algorithm to retrieve the tracks, i.e. an end to end machine learning approach for tracking. A supervised mapping model was defined and tested with the results showing a significant improvement in same particle hits grouping especially for high P_T tracks. A KMeans clustering algorithm retrieved the particle trajectories in the mapped buckets. This technique loses up to 20% of tracks in the central pseudorapidity region due to the limitation of the KMeans clustering algorithm in disentangling merging tracks. While the majority of tracks have been successfully found with this approach, an important improvement is the design of a custom clustering algorithm that takes into account tracking constraints.

The next step is to challenge this technique with a simulation dataset and more specifically the ATLAS Phase-II Inner Tracker (ITk) simulation dataset. The fast search is evaluated on the ITk dataset in Chapter 8 while a new metric learning approach combined with the fast search is proposed in Chapter 9. A novel tracking inspired clustering algorithm is also proposed in Chapter 9.

Bibliography

[1] Amrouche, Sabrina, et al. "similarity search for charged particle tracking." 2019 IEEE International Conference on Big Data (Big Data). IEEE, 2019.

[2] Amrouche, Sabrina, et al. "Hashing and metric learning for charged particle tracking."

8

Similarity search with ATLAS Phase-II Inner Tracker

8.1 Introduction

Applying similarity search on an ITk dataset presents a multitude of additional challenges as opposed to the "toy" detector used in the TrackML challenge. Not only the ITk optimized geometry is different (with more layers for example) but also the simulated samples contain the difficulties encountered in the real world experiments. By design, the granularity of the detector allows only a low resolution on the inner angles of the clusters (introduced in Section 1.3) thus penalizing the metric learning model that was successful with the TrackML dataset.

In this chapter, we will firstly review all of the characteristics of the ITk dataset and their technical implications on the techniques proposed in Chapter 7. The application of ANN techniques is then demonstrated with a performance analysis of the ATLAS standard tracking restricted to the buckets.

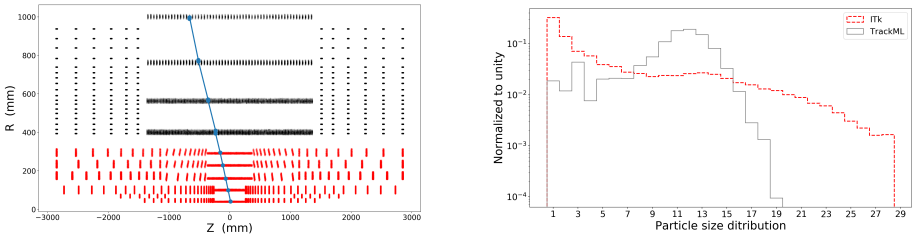
8.2 The ITk simulation dataset

The studied events result from the simulation of a top pair ($t\bar{t}$) production with 200 overlaid pile-up (similar physics to the TrackML challenge). The standard tracking reconstruction is ran on the events and provides a *reconstructed* label for each hit. The label from simulation will be referred to as *truth*. The ITk detector (presented in section 2.2.3) has two sub components : pixels and strips. The collections of hits created by an event is displayed in Figure 8.1(a) along the R-Z axes with pixels and strips shown in different colors. An example particle is also highlighted on the plot with its hits represented by dots on the detector modules. The ITk dataset is simulated using Geant4 (see Chapter 2.3) and contains detector effects such as charge sharing.

Table 8.1 summarizes the relevant characteristics of an ITk event compared to the TrackML. These dataset characteristics were chosen because of their impact on the proposed solution of similarity search. For example, we choose to compare the number of innermost pixel coordinates between the two datasets because of its significant impact on the finding of particles with an offset along the beam axis. This is further discussed in Section 8.5.1. In the TrackML dataset, the particles present in the dataset are also for target of a model, i.e. simulated tracks = reconstructable tracks. In other words, we are interested in all the particles simulated in the dataset. In the ITk samples however, additionally to the truth information obtained through simulation, it is possible to run the ATLAS standard tracking to obtain only the fraction

of *interesting*[1] particles. In the case of the ITk samples then, the total reconstructed (reco) particles is the subset of total simulated (truth) particles that pass standard cuts. The major difference with the TrackML dataset is the number of hits per event. A typical event has on average 400K measurement which is at least a factor of 4 increase compared to TrackML. The differences in the total number of hits and also in the number of hits per particle (particle size) are the result of different detector geometries. The majority of particles in the ITk sample are noise particles with less than 4 hits. These noise particles are filtered out in the TrackML dataset to maintain the noise level at an average of 10% and thus simplify the reconstruction for participants. This explains the wide difference in noise levels between TrackML and ITk.

The distribution of particle sizes in log scale for both ITk and TrackML is presented in Figure 8.1(b). As will be discussed further, this size difference will impact the bucket size definition as well.



(a) (b)

Figure 8.1: (a) Longitudinal view of the ITk layout with pixels in red and strips in black (only the center is shown). A particle is shown crossing the different layers (9) of the detector. (b) Particle size distribution in log scale for the ITk dataset. The TrackML particle size distribution is highlighted in grey for comparison.

Detector	$<\mu>$	Total measurements (Pixels)	Average particle size	Total Truth particles	Total Reco particles	Innermost pixel z $>250mm$	Noise level
ITk layout	200	400K (230K)	15	15K	2K	7K	53%
TrackML	200	100K	10	10K	/	4K	11%

Table 8.1: Summary of relevant differences between ITk and TrackML events.

The hit merging (Section 3.1) which was not simulated in the TrackML dataset is also an additional challenge. In the ITk dataset, hits are no longer associated to a unique particle but rather have a variable list of particle barcodes attached to them. On average, **8%** of hits are associated to more than one particle.

Moreover, the ITk pixel cluster shape resolution being different, the inner angles that are derived from ToT[2] have also a lower resolution. This loss primarily impacts the metric learning model that relies on the inner angles to discriminate between tracks. The distribution of the ITk cluster shapes is shown in Figure 8.2 while the TrackML one is highlighted in grey for comparison. The two distributions are normalized with a log scale to emphasis larger clusters. Although both datasets are dominated by one and two pixel clusters,

[1]Depending on the physics targeted, threshold such as P_T can be increased/decreased. This heavily affects the total number of reconstructed particles.

[2]In TrackML, the inner angles are derived from the charge and evaluated using the path of the particle.

the ITk layout produces fewer clusters larger than 3. This is a major difference as the angle definition becomes relevant starting from 3 pixels, i.e. a two (or one) pixels cluster shape cannot define an inner angle for the passing particle. As a consequence, most close-by tracks have pixels with similar inner angles. Although the resolution of cluster shapes is lower, the shape features, however little, can be extracted with a Convolutional Neural Network (Section 8.8).

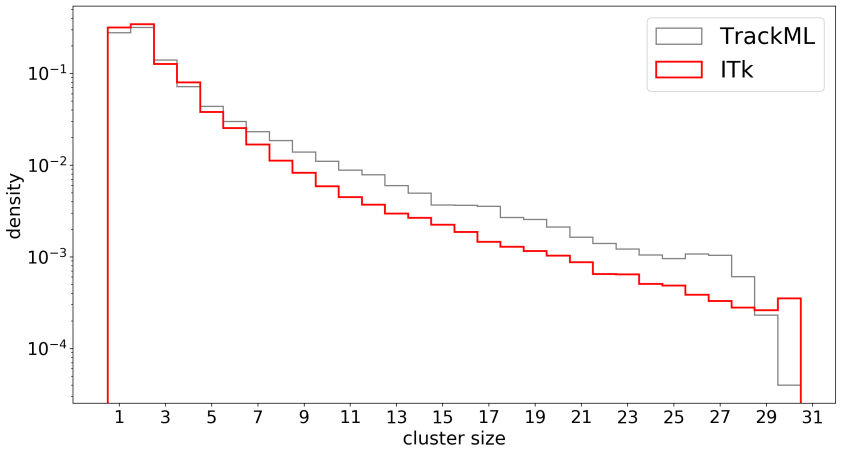


Figure 8.2: Cluster sizes (pixels) distribution for the ITk and the TrackML dataset. The two distributions are normalized due to dataset size differences.

The different characteristics of the ITk dataset are reflected in the building of ANN buckets as well as on the application of metric learning. In the next section we will analyze the content of ANN buckets derived from the ITk. These buckets will then be used as containers to run the ATLAS standard tracking on.

8.3 ANN buckets on the ITk dataset

Similarly to the ANN based algorithm presented in Chapter 7, hits of an event are indexed using an angular similarity on their 3D coordinates (x, y, z). An ANN bucket of size 50^3 is constructed for every hit in the dataset. Each bucket is further investigated to retrieve the full particle trace (if any). Annoy library is used to construct and query from the ANN index. Figure 8.3 shows the distribution of leading particle sizes per bucket for one event. In the background, the same metric applied to the TrackML dataset is shown for comparison. Although both distributions peak at 4 hits, buckets from ITk contain considerably shorter tracks. This is a direct consequence of the noise level (introduced in Section 7.1) in the ITk dataset. Indeed, while the TrackML dataset contains only 11% of noise particles, the ITk has more than 53% of noise particles. In fact, these particles are not reconstructed by the standard ATLAS tracking since they do not pass standard cuts on energy and/or P_T. Low momentum or displaced particles are reconstructed in a later iteration of the tracking algorithm with looser cuts.

It is worth noting that since a bucket is constructed for every hit, the overlap between these 50-hits buckets is non negligible. A particle can be contained in more than one bucket and a filtering strategy is therefore necessary.

[3]The bucket size is a model parameter and is chosen to maximize the probability of containing a track.

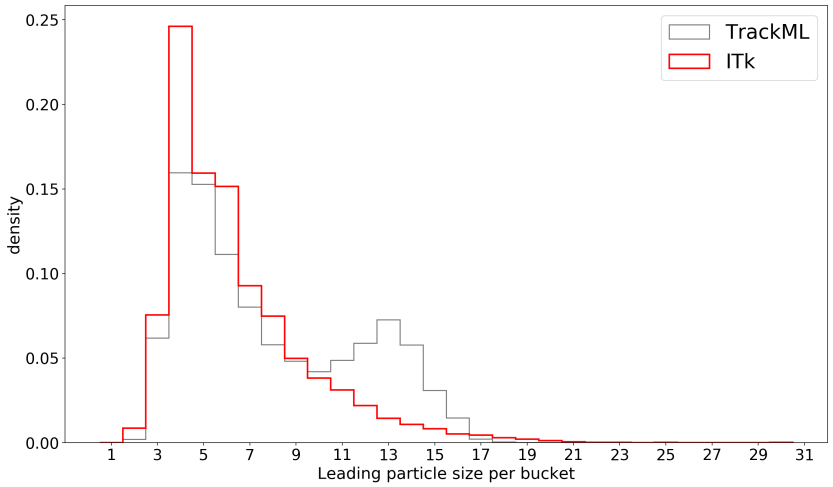


Figure 8.3: Distribution of the largest particle size per bucket for the ITk dataset. The TrackML distribution is highlighted in grey for comparison.

Bucket overlap studies are presented in Section 8.7.

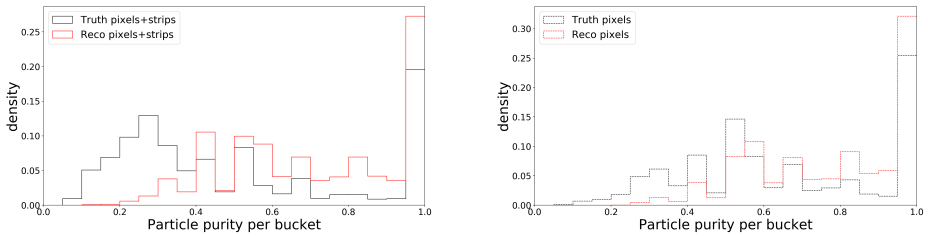
In the context of tracking, an ANN bucket is built to reduce the combinatorial complexity of tracking by increasing the probability of finding a track (ideally complete) within this small collection of hits. In order to determine the maximum attainable efficiency of the ANN buckets, we have to examine the fraction of particles contained in buckets. A particle can be fully contained in a 50 hits bucket or only partly with some hits *escaping* the cone shaped bucket especially in case of long or highly curved tracks. This particle **purity** is computed from every bucket in an event and only the maximum purity per particle is kept. The particle purity, in this case, is defined as the number of particle hits contained a bucket divided by the total number of hits produced by the particle. For example, if the complete trace of particle is fully contained in an ANN bucket, its corresponding purity is 1. Ideally, all the particles are contained in a buckets but since many physics effects distort the track shape and the ANN buckets are built from fixed angular distances, the particle purity ranges from 20% to 100%.

Figure 8.4 shows the particle purity distribution per event. The purity is computed considering all the possible buckets of an event, i.e. every hit is queried. We are interested in different purity distributions :

- Purity of reconstructable particles (given the ATLAS standard tracking) in the full detector. The distribution per event is shown in red in Figure 8.4(a).
- Purity of all simulated particles in the full detector. The distribution per event is shown in black in Figure 8.4(a).
- Purity of reconstructable particles in the pixel sub-detector. The distribution per event is shown in red in Figure 8.4(b).
- Purity of all simulated particles in the pixel sub-detector. The distribution per event is shown in black in Figure 8.4(b).

Looking at the pixel purity provides additional information on the seeding potential of ANN buckets. Figure 8.4(a) shows the distribution of the full

particles against truth and reco while Figure 8.4(b) shows the same quantities restricted to the pixel sub-detector.



(a) (b)

Figure 8.4: Maximum particle purity distribution in ANN buckets on the ITk dataset.

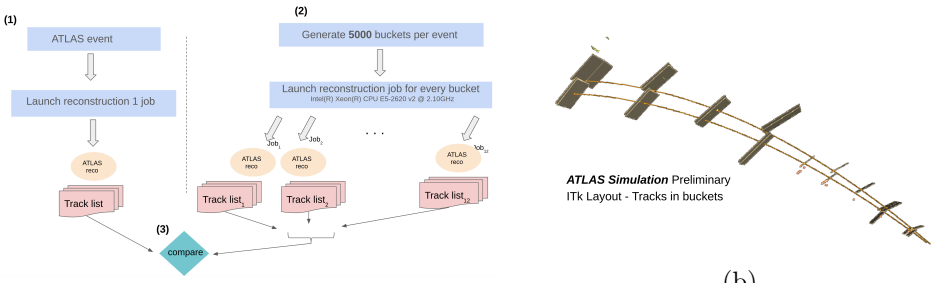
Using the standard purity cut of 80%, the potential reco efficiency on the full detector is of **40%**. This means that assuming a perfect tracking algorithm per bucket, 40% of particles will be retrieved in ANN buckets of 50 hits. This shows the important difference between the TrackML dataset and the ITk where, in TrackML, more than 90% of particles were contained in 50 hits buckets. In Figure 8.4(b) we can see that although only 40% of the distribution fills high bins, the remaining purity values are well above 50%. This means that the buckets still contain an important part of the track : a seed. In the next section (8.4), the first scenario is explored where the ATLAS tracking algorithm runs on each ANN bucket independently. In section 8.5, the ANN buckets are only used for seeding and the ATLAS tracking algorithm completes the found seeds with hits from the full event.

8.4 Standard ATLAS tracking in buckets

Although the ATLAS standard tracking algorithm was not designed to run on very small bins, we show in this section the feasibility of running the full algorithm in buckets of 50 hits. More specifically, the tracking algorithm, in every stage, has only access to the hits in the bucket. Figure 8.5(a) describes the strategy adopted to compare both reconstructions. Technically, multiple (5000 as an initial test) reconstruction jobs are launched in parallel and every job has a bucket of 50 hits as input. The tracks reconstructed in buckets are then compared to the same tracks found in the full event through the track lists that are produced.

Figure 8.5(b) shows a display of a 50 hits buckets and the two tracks found in it through the ATLAS tracking algorithm. This is the output of a single job described in the second step of the pipeline in Figure 8.5(a). The surfaces shown are the modules that recorded the passage of the particle. The remaining hits that were not used to build a trajectory are shown by the yellow dots. In this case, the ATLAS tracking algorithm runs on the hits of the bucket including seeding, track finding and ambiguity solving (see Chapter 3).

It is expected that the standard tracking produces the same tracks when considering similar input hits. When a track is found in a *traditional* full event reconstruction and also when the tracking is restricted to 50 hits buckets, then this track is referred to as **matched**. For the purpose of performance comparison, we consider that a track is matched when both reconstructions



(a) (b)

Figure 8.5: (a) Standard ATLAS tracking restricted to ANN buckets strategy.(b) Illustration of ATLAS reconstruction in a bucket of 50 hits [1]. The picture shows the measurements (hits) in the bucket with the two reconstructed tracks (continuous lines). The surfaces shown are associated to the active sensors crossed by the reconstructed tracks.

(full event and bucket) produce at least seven[4] hits in common. If the bucket contains all of a particle hits, it is expected that we find the same track whether inside a bucket or on the full event. In Figure 8.6, the sizes of matched tracks between the bucket reconstruction and a full event are compared. Every pixel in the 2d histogram is a matched track with its horizontal axis coordinate indicating the size of the track in the full event and its vertical axis coordinate the size of the corresponding track found in a bucket. The colors indicate the fraction matched per sizes. If every bucket contained a track fully, this plot would be a bright diagonal. The pixels below the diagonal are tracks that have more hits when the reconstruction is run on a full event, i.e. the bucket allowed only a partial reconstruction. The tracks above the diagonal, on the other hand, have interestingly *more* hits in the bucket compared to their full event twins. This is explained by the higher purity of the bucket environment that significantly reduces combinatorics (noise) and allows better combinations to be tried (and selected).

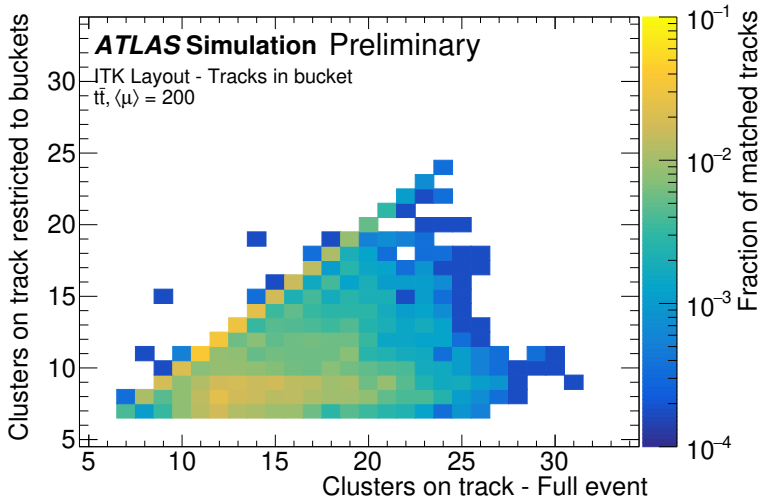


Figure 8.6: Matching of tracks between a classical full event reconstruction and a reconstruction restricted to buckets of 50 hits

In order to get a sense of the tracking performance when the track finder has access to complete traces, a truth extension is applied at the bucket level.

[4]If two tracks share 7 hits, they both describe the same particle.

Buckets of at least four hits from the same particle are extended to contain the full traces (adding the particle hits). The obtained buckets have variable sizes since multiple seeds could be completed in a single bucket. Figure 8.7 shows the sizes of tracks found inside buckets and compares them to matched tracks in truth extended buckets. Since the truth extended buckets produce complete traces (similar to full event reconstruction), Figure 8.6 and Figure 8.7 show the same patterns indicating that tracks found in truth extended buckets are comparable to the tracks found in the standard full event reconstruction.

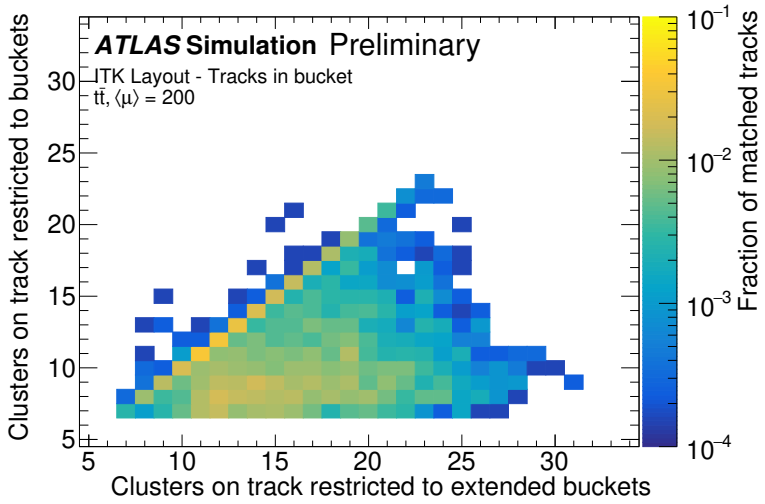


Figure 8.7: Matching of tracks between truth extended buckets reconstruction and buckets of 50 hits

Since the buckets (without extension) do not fully contain particle traces in the ITk dataset, an alternative is to allow the track finder to complete the missing hits by accessing the full event. In other words, use the buckets only at the seeding stage and from there run conventional track finding and ambiguity solving.

8.5 Standard ATLAS seeding in buckets

In this section, the ATLAS standard tracking algorithm runs the seeding procedure in ANN buckets and is allowed to complete a found track from the full event. Since only the seeding is performed inside the bucket, we consider only pixel hits to build the buckets. First, an ANN index is built for every event then buckets are sampled from the index and an instance of the tracking algorithm is run on every bucket independently. The tracks found across the different jobs are combined and form the output of the tracking pipeline : a list of particle trajectories. The overall approach is summarized in Figure 8.8.

Given a hit, its associated bucket is defined as the set of hits closest to it using an angular distance. By building a sufficient number of buckets across the detector, every track seed is contained in at least one bucket. Seeds are built from hits contained in the buckets and then completed by the track finder with access to hits from the full event. If a bucket does not contain any valid seed, no track candidate and therefore no output track is produced from that bucket.

A bucket may contain a large number of hits from the same particle. While only part of these hits are used to form seeds, the remaining hits are used in the track finding phase to complete the seed into track candidate.

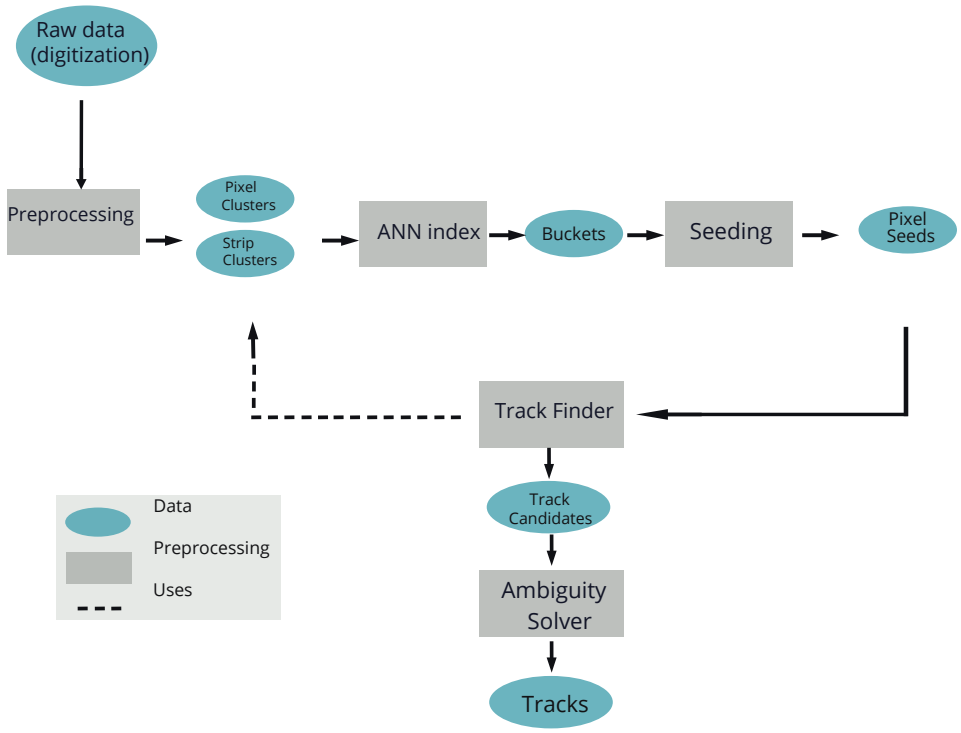


Figure 8.8: Proposed pipeline to run the standard ATLAS tracking algorithm with seeding in buckets. The ANN index is built on pixel clusters only. Fixed size buckets are sampled from the index and passed to the seeding algorithm. Seeds are formed by 3 space points compatible with a helical track model that passes a min P_T requirement, a max d_0 assumption and a 4th layer confirmation, i.e. 4 space points to form a seed. The Track Finder, followed by the Ambiguity Solver return a final track as described in Chapter 3.

8.5.1 Bucket sampling strategy

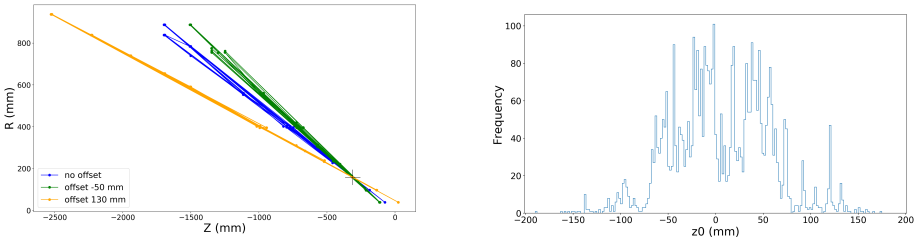
Although building buckets from each hit in an event guaranties to find all[5] track seeds, it is not interesting to do so since only a very small fraction of them will eventually produce particles. It is therefore necessary to *select* the buckets on which to run tracking. Firstly, we present a random selection in different pseudorapidity bins that is later filtered using a machine learning based classifier.

Multiple index search

Throughout this document, the ANN index was described with 3D points coordinates and an angular distance. However, assuming a fixed origin for the buckets naturally fails to contain displaced tracks. The beam spot spreads to $+/-$ 20cm along the z coordinate and produces tracks that cannot be contained with buckets centered at (0,0,0). Figure 8.9(a) shows a projection in the (R,z) plane of three different bucket origins. We refer to buckets with a non zero vertex as *displaced buckets*.

An ANN index solely built on an angular distance (pointing to the origin

[5]Considering standard P_T cuts



(a) (b)

Figure 8.9: (a) Displaced buckets in z coordinate with three offsets illustrated. Hits are connected by lines within a bucket for better visualization. The black cross denotes the query hit position. (b) Reconstructed z_0 distribution.

z=0) can produce buckets that contain tracks with a maximum displacement of 50mm. This maximum threshold was determined after running the standard reconstruction on non displaced buckets only and analyzing the z_0 distribution of found tracks. It is then necessary to cover the whole z_0 range shown in Figure 8.9(b) by building multiple indexes with different z_0 offsets. Technically, this is achieved by building in parallel different ANN indexes each on a shifted z coordinate. The contribution of these displaced indexes is weighted according to the actual distribution of z_0 in an event. For example, indexes with an offset<70mm have a higher weight than those at an offset>70mm since the proportion of tracks with $z_0 < 70$mm is much higher. These weights are also constructed according to the eta η distribution of an event.

8.6 Reconstruction result analysis

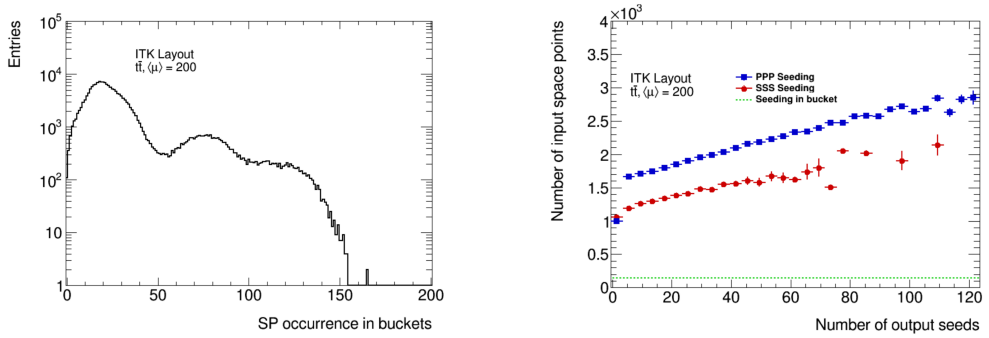
When restricting the seeding to fixed-size buckets of 70 hits, we are interested in the following metrics:

- The fraction of seeds that form track candidates and later accepted tracks, i.e final reconstruction output.
- The time spent per bucket for :
  - The seed creation.
  - The building of the track candidate from an accepted seed.
  - The ambiguity solving
- The quality of found tracks :
  - The size of a track formed from a seeding inside buckets as opposed to its matched track formed with seeding in the full event.
  - The agreement of track parameters in the matched tracks ensemble.
- The event final reconstruction efficiency and the impact point resolution comparison.

In the remainder of this chapter, the enumerated metrics are presented and discussed.

In every ANN bucket, the ATLAS seeding procedure, through combinatorics and minimal thresholds, extracts a number of pixel seeds. This number varies depending on the quality of the bucket and on the cuts applied in the

queried region. Figure 8.10(a) shows the distribution of the number of space points (SP) used to construct the seeds. As a reference, the standard seeding requires few thousands SPs. The illustrated distribution shows two important bumps at 20 and 70 and very few to no buckets that produce more than 150 SPs. Each track seed is formed by combining three space points from the bucket hits (or from the full event in the standard reconstruction). The total number of input space points that are considered to form seeds is illustrated as a function of the number of output seed in Figure 8.10(b). Whereas in the standard reconstruction thousands SPs are required to form seeds, only a maximum of 100-120 SPs are in the case of seeding in the bucket.



(a) (b)

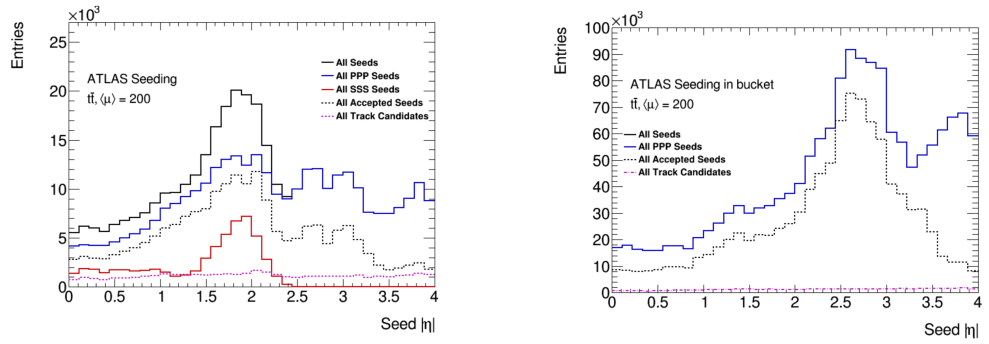
Figure 8.10: (a) Distribution of the number of space points (SPs) used per bucket to form seeds. (b) Evolution of the number of SPs as a function of output seeds. A comparison is presented by seeding in the full event (pixel PPP and strip SSS seeds) and seeding in buckets.

The pure environment of a bucket offers the possibility of building a high number of seeds with a constant number of space points. This is illustrated by the flat green line in Figure 8.10(b). In standard reconstruction, the same number of seeds is achieved through an important increase of the number of SPs. The linear increase in the standard reconstruction is also due to a non static bin size. The bins used in standard tracking are formed using a grid in (z, φ) allowing the size of the bin to vary from few dozen points to few hundreds (up to 800 on average).

As mentioned earlier, the number of seeds varies depending on the region of the detector and more specifically depending on the eta region considered. The distribution of the number of seeds as a function of the pseudorapidity in standard reconstruction is presented in Figure 8.11(a). As a contrast, the seeding in buckets equivalent is highlighted in Figure 8.11(b). The two figures are obtained from running on the same data samples. Only the pixels (PPP) are considered when building the seeds from buckets resulting in the similar distributions for "All Seeds" and "All PPP Seeds".

From Figure 8.11(b), we can see that the two reconstruction variants populate different pseudorapidity regions for PPP seeds. The fraction of track candidates is similar between the two approaches. The total number of seeds formed in buckets is noticeably higher than in the standard reconstruction and this mainly caused by the absence of an overlap removal between buckets (discussed in Section 8.7). Additionally to an overlap removal, an automatic classification of buckets into "contains a track" and "does not contain a track" categories has the potential to significantly reduce the number of buckets considered and therefore of seeds (discussed in Section 8.8).

One of the main motivations in favor of an ANN based seeding is the potential time improvement. We are mainly interested in timing the following



(a) (b)

Figure 8.11: Distribution of the number of seeds as a function of the seed pseudorapidity η. In (a), the seeds distribution in the standard reconstruction is highlighted with pixel and strip seeds as well as the passage from seeds to accepted seeds and to track candidates. The same quantities are shown in (b) with the seeding restricted to buckets. Seeds are formed from pixel hits only, i.e. no strips only Seeds (SSS).

algorithms per buckets:

- Seed production execution time.
- Track Finding execution time.
- Ambiguity Solving execution time.

These execution times depend on the input size of each algorithm. For example, the Track Finder execution time depends on the number of seeds in input and the ambiguity solver depends on the number of track candidates in input. The seed production time depends on the number of seeds produces and implicitly, the bucket quality. These execution times are represented in Figure 8.12.

Figure 8.12(a) shows the evolution of the seed production time, per bucket, as a function of the number of output seeds (green distribution). As a reference, the standard seeding execution time on pixels and strips is highlighted (blue and red distributions). It is interesting to see that for the same number of output seeds, the seeding in buckets is much faster than the standard seeding. Moreover, the seeding in buckets results in a nearly flat distribution that maintains similar execution times independently of the number of seeds produced. This is a direct consequence of the effects shown in Figure 8.10(b), i.e. fixed-size buckets (SPs) allow a nearly constant stream in output seeds and therefore a constant seeding time.

After the seeds are created, the track finder builds trajectories from each seed by adding hits from the full event (not only the bucket). Due to the high number of seeds and the combinatorial nature of the track finder, the process of building track candidates is slow. Figure 8.12(b) shows the potential speedup of running the track finder on seeds created in an ANN bucket. The execution time of the track finding algorithm is displayed as a function of the number of pixel seeds. The scaling is linear and the higher the number of seeds, the longer it takes to building track candidates from each one of them. It is interesting to note that since the seeds are formed from buckets of 70 hits, the maximum number of seeds considered per reconstruction job is 60 on average as opposed to 120 in standard reconstruction. Figure 8.12(c) shows the distribution of the number pixel seeds and the number of resulting track

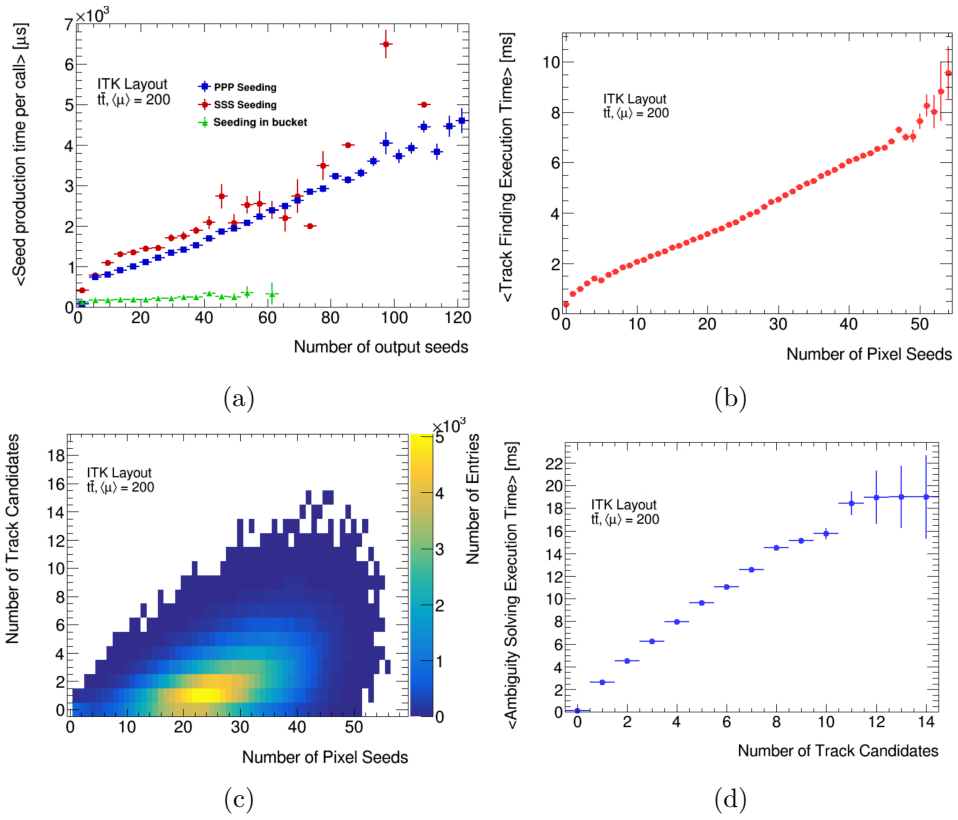


Figure 8.12: (a) Production time of the seeding. (b) Track finding time as a function of the number of pixel seeds in a bucket. (c) Number of track candidates as a function of the number of pixel seeds. (d) Ambiguity solving time as a function of the number of track candidates per bucket.

candidates. The majority of buckets will produce on average 25 seeds and the track finder spends an average of $3-4$ms per bucket for the construction of track candidates. The somewhat diagonal shown in Figure 8.12(c) illustrates the conversion of pixel seeds into track candidates. In the majority of cases only 1 candidate is formed from the seeds of the bucket. More seeds tend to produce more track candidates and few thousands buckets do not produce any track candidate. This is also directly related to the minimal acceptance thresholds considered when building a track. Indeed, from the buckets that do not produce any track, an important fraction contains seeds of low momentum particles.

Figure 8.12(d) shows the time spent in ambiguity solving per track candidate number. The dependence is also linear and more track candidates directly translates into more time spent in the ambiguity solving. As mentioned previously, the wide majority of buckets produce 1 track candidate and it takes on average 3ms for the ambiguity solver to reject or accept this track. Most of time this track is accepted.

The mapping between track candidates and final tracks (output of the ambiguity solver) are shown in Figure 8.13. The majority of buckets produce one track candidate that is accepted. The second most common cases are buckets with a single track candidate that is rejected by the ambiguity solver or two track candidates with only one that is accepted. It is interesting to see the linear correlation between track candidates and accepted tracks. Some buckets (few dozen) produce up to 9 track candidates and up to 8 of them are accepted to make up the final reconstruction output.

Now that we have evaluated the time and the quality of the seeding, the

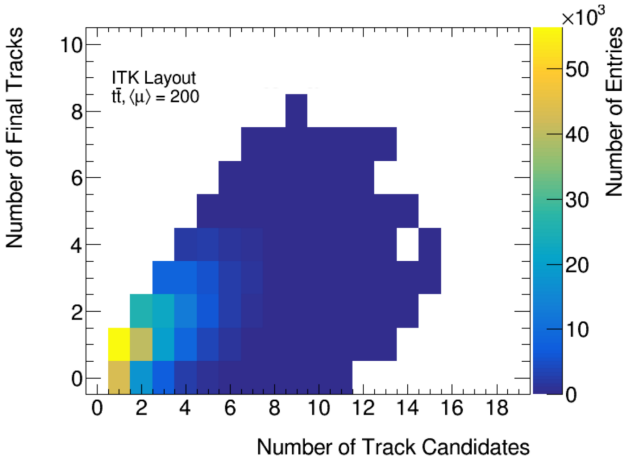
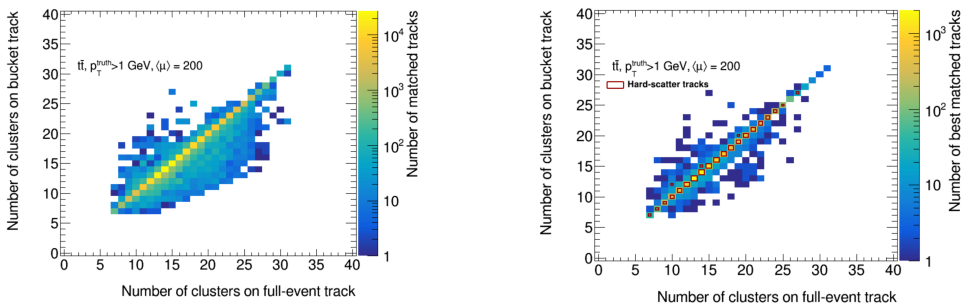


Figure 8.13: Number of final tracks as a function of the number of track candidates per bucket.

track finding and the ambiguity solving in ANN buckets, we can analyze and compare the final output of these algorithms : particle trajectories.

A particle trajectory typically contains 20 hits. Among these, 4 hits where used as a seed to build the initial track hypothesis (used in the track finder). Depending on which 4 hits where used as a seed, the final track content might slightly vary. We are now interested in comparing the size of a track found in standard reconstruction with the size of a matched[6] track found with a seeding restricted to ANN buckets.

Figure 8.14 shows the distribution of the sizes of matched tracks between the standard reconstruction and the buckets seeding reconstruction with a minimum P_T cut of 1GeV. For every bucket that yields a track, a matching is done with the standard reconstruction output tracks. If they share at least 7 hits, both their sizes are recorded. The horizontal axis of Figure 8.14 shows the sizes of final tracks from the standard reconstruction and the vertical axis shows the matched tracks from seeding in buckets. If a track is reconstructed in multiple buckets, all its sizes are recorded in Figure 8.14(a). In Figure 8.14(b) however, only the longest matched size is recorded.



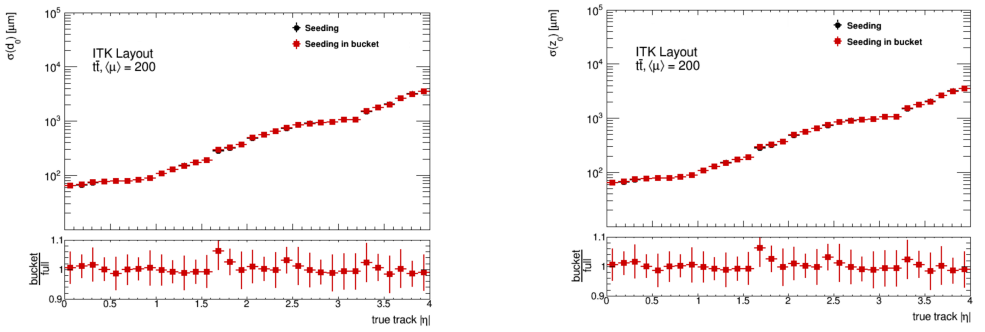
(a) (b)

Figure 8.14: Matched tracks length comparison between standard full event reconstruction and seeding inside buckets reconstruction. (a) shows the matching between all reconstructed tracks with at least 7 hits while (b) shows only the matching between a track found in the standard full event reconstruction and its best (longest) match with the seeding restricted to buckets. In (b) tracks coming from the hard-scatter vertex are highlighted in a red square.

[6]We assume that two tracks are matched if they share at least 7 hits. Matched tracks produce similar track parameters.

Hard-scatter tracks (most energetic and most interesting), denoted with red squares in Figure 8.14(b), populate the diagonal, i.e. tracks from the hard scatter vertex are found similarly in the standard reconstruction and in the bucket seeding reconstruction. The tracks above the diagonal contain more hits when built with a seed from a bucket than with a seed from standard reconstruction. This suggests that those bucket seed allow a better resolution of the tracks. Matched tracks below the diagonal result from the opposite mechanism where a seed from the standard reconstruction chain provides higher resolution compared to a seed built in a bucket.

Since the hit content of the tracks returned from a seeding inside buckets are not exactly similar to the ones returned from standard reconstruction, a comparison of impact point (IP) resolution parameters is necessary. This comparison is between the d_0 and z_0 of the vertices in each of the two reconstructions.



(a) (b)

Figure 8.15: IP resolution along d_0 and z_0 as a function of the pseudorapidity η between standard reconstruction (ATLAS Seeding) and seeding restricted to buckets reconstruction (ATLAS Seeding in bucket).

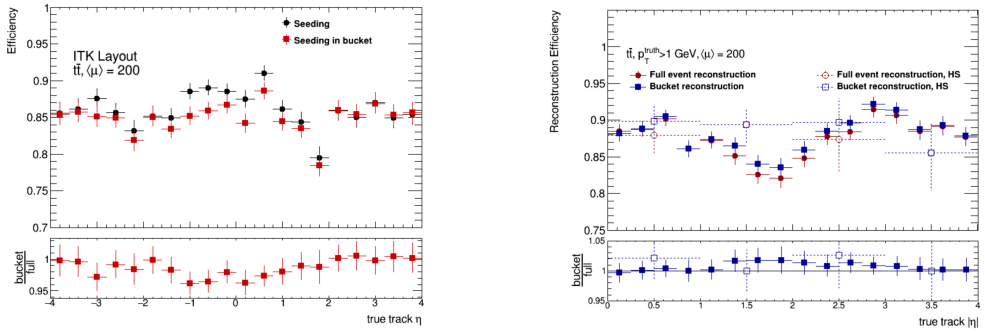
Figure 8.15 shows the $\sigma(d_0)$ and $\sigma(z_0)$ evolution as a function of the track pseudorapidity η. The ratios are also shown in the figures. Apart from slight variations, the two set of tracks are in agreement with few percent and provide very close IP resolution values.

The final and most important item of comparison is the event efficiency. The reconstruction efficiency is the fraction of tracks that is successfully found from the total number of simulated truth tracks. Truth tracks are stable, charged and with P_T>1GeV. The efficiency of the two reconstructions is shown in Figure 8.16(a). The efficiency ratio is shown at the bottom of the figure. The seeding inside buckets is slightly less efficient in some pseudorapidity regions with an average of 5% inefficiency per event compared to the standard reconstruction. The lost tracks (inefficiency) appear to be mostly located in the central region ($|\eta| < 2$). Note that the standard ATLAS efficiency is around 88%.

The standard reconstruction is optimized to run on the ITk layout. This optimization implies tighter thresholds to reduce combinatorics (see default thresholds in Section 2.2.3). In the bucket environment however, the number of hits being always 70, will allow the production of seeds that do not pass the acceptance thresholds. The inefficiency seen in Figure 8.16(a) is primarily due to these tight constraints on the seeding.

To confirm this hypothesis, we loosen up two seeding cuts : The minimum distance between hits for PPP seeds and the maximum seed d_0. Since an ANN bucket is agnostic to the detector layout, the distance between two hits can be arbitrary small and the seed d_0 arbitrary large as long as the hits within a

bucket are the closest neighbors in terms of the angular distance.



(a) (b)

Figure 8.16: Global efficiency of seeding in buckets reconstruction compared to the standard ATLAS reconstruction. (a) uses standard seeding thresholds for both reconstructions while (b) shows the impact of relaxing the seeding thresholds when running inside buckets. The standard reconstruction in (b) still uses the standard thresholds.

The changes on the two mentioned seeding parameters are summarized in Table 8.2. This loosening of the seeding thresholds will not allow more seeds to be accepted but rather more seeds to be considered. The bucket environment will therefore be less penalized when containing close-by hits or larger d_0 seeds. These new threshold values were chosen to minimize zero seeds buckets. The resulting efficiency with this relaxed seeding strategy is shown in Figure 8.16(b). Note that the standard ATLAS reference in the Figure still uses standard seeding cuts (not an apple to apple comparison). The standard seeding thresholds were maintained as a reference to ensure an optimized reconstruction, i.e. looser cuts in the standard reconstruction (full event) will result in a combinatorial explosion and a performance loss (that is why these cuts were chosen in the first place).

Seeding	Min hit distance	Max d_0 seed
Standard Seeding	6 mm	2 mm
Relaxed Seeding	0.1 mm	5 mm

Table 8.2: Summary of the change to the seeding thresholds.

The bucket reconstruction efficiency in Figure 8.16(b) is now much higher and in fact higher than the standard full event reconstruction although this is only used as a marker and we are interested in the overall truth efficiency, i.e. fraction of found tracks from the total simulated. The hard-scatter efficiency is also highlighted for both reconstructions. We can clearly see that the average event efficiency is now 90% instead of 88% and that especially in the $1 < |\eta| < 3$ region the ratio of reconstructed tracks is much higher. Moreover, the relaxed seeding cuts allow to find relevant tracks of $P_T > 1$GeV from the hard-scatter event. In Figure 8.17 we can see the spread of the reconstruction tracks along P_T, d_0 and z_0.

Generally, by relaxing the seeding inside the buckets we are able to retrieve more tracks of different transverse momentum (P_T) and offsets (d_0,z_0). The high p_T spectrum has less statistics and as a result no clear conclusions can be derived for the efficiency at 4GeV for example. The same can be said for high d_0 and z_0 offsets where we can see an increase in uncertainty. The additional tracks were found from exactly the same buckets that gave a lower efficiency in

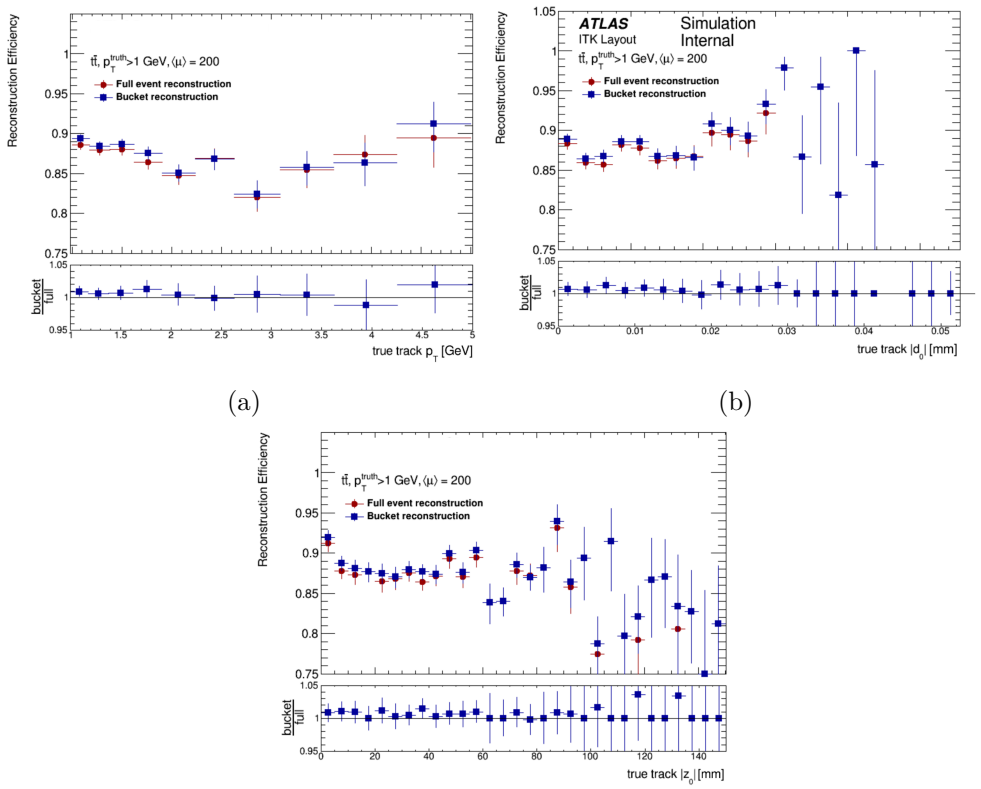


Figure 8.17: Efficiency of relaxed seeding in buckets as a function of P_T (GeV), d_0(mm) and z_0(mm).

Figure 8.16(a). This means that although the seeding acceptance thresholds were lowered, the number of processed buckets is the same.

Unmatched tracks

Additionally to the improved overall performances enabled by a relaxation in the seeding cuts, the bucket based approach finds more tracks than the standard ATLAS reconstruction (that uses stricter cuts). These additional tracks are called *unmatched* because they do not have a corresponding track found in the standard reconstruction and yet they are simulated and thus associated to a truth particle. It is worth mentioning that even when using similar cuts between the standard reconstruction and the seeding in buckets, additional unmatched tracks are found.

The fraction of unmatched tracks is overlaid in the efficiency plot of Figure 8.18 as a function of the pseudorapidity. The majority of new tracks are found in the $1.5 < |\eta| < 3$ region. These unmatched are primarily recovered because of the high purity of the bucket environment that allows the creation of more valid hit combinations.

When an unmatched track is found by the bucket seeding reconstruction, two cases arise:

- The track is matched to a true particle, i.e. a link to an existing particle can be retrieved.

- The track is not matched to any true simulated particle, i.e. it is unlinked to simulated particles and therefore it is a fake track. We call these tracks *unlinked*.

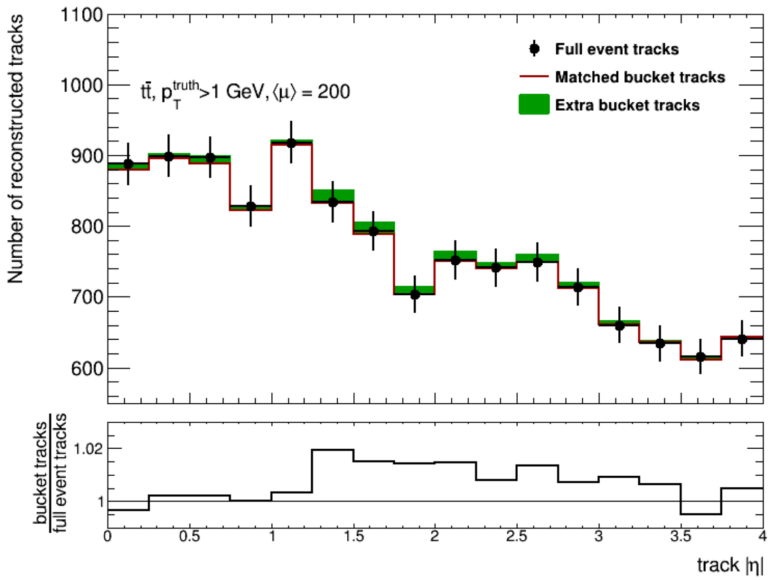


Figure 8.18: Extra tracks (unmatched) found by restricting and relaxing the seeding in ANN buckets. The full event tracks represent those found with standard ATLAS reconstruction. The track count ratio per eta bin is also illustrated.

The unmatched linked tracks (true extra tracks) span across the whole P_T spectrum but the majority have a $P_T < 2$GeV as shown in Figure 8.19(a). The unmatched unlinked tracks (fakes) are very rare and their track parameters are irrelevant since they are non real particle trajectories. They can however bias the total reconstructed parameters. The fraction of unlinked tracks is shown in Figure 8.19(a). The unmatched tracks also span across the full d_0 and z_0 ranges as presented in Figures 8.19(b) and 8.19(c).

There is a simple criteria to distinguish between fake tracks (unlinked) and valid tracks. Unlinked tracks have a maximum of 9 hits on average whereas linked tracks have more than 10 hits as illustrated in Figure 8.20. Unlinked tracks also tend to populate the forward detector region.

In summary, restricting the seeding to 70 hit buckets achieves an overall efficiency that is 5% lower than the standard ATLAS reconstruction when using standard seeding cuts and 1-2% higher when relaxing those cuts. In both cases, we see additional unmatched true tracks with different transverse momentum and z_0, d_0 offsets. The standard tracking needs on average 4ms to find a track in a 70 hit bucket. This process can be massively parallel (on GPUs for example) since the buckets are queried independently. Two aspects, however, need further consideration :

1. The overlap between buckets: Querying independent hits in the ANN index can produce heavily overlapping buckets.

2. Noise buckets : an important fraction of buckets do not produce any accepted seed.

8.7 Bucket overlap analysis

If two close-by hits are queried, the resulting ANN buckets might share some of their hits. An example of these cases is presented in Figure 8.21 where two 20 hits ANN buckets share 12 hits. The query hits, denoted by the arrows

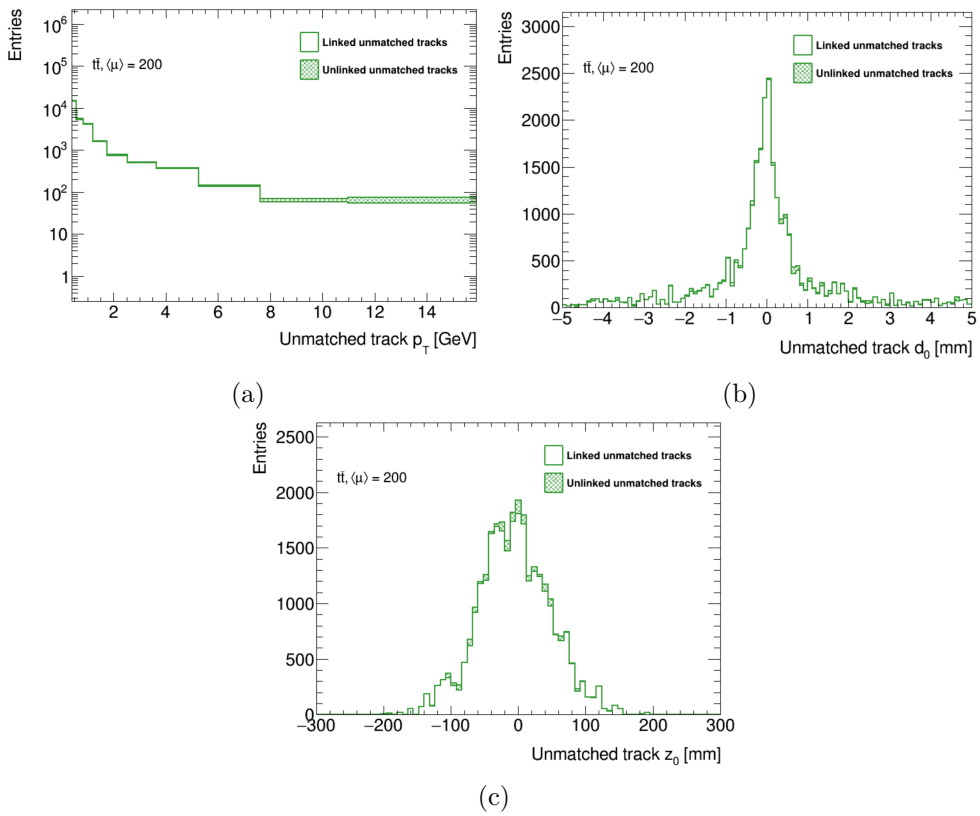


Figure 8.19: Parameters of the unmatched and unlinked tracks averaged over 10 events.

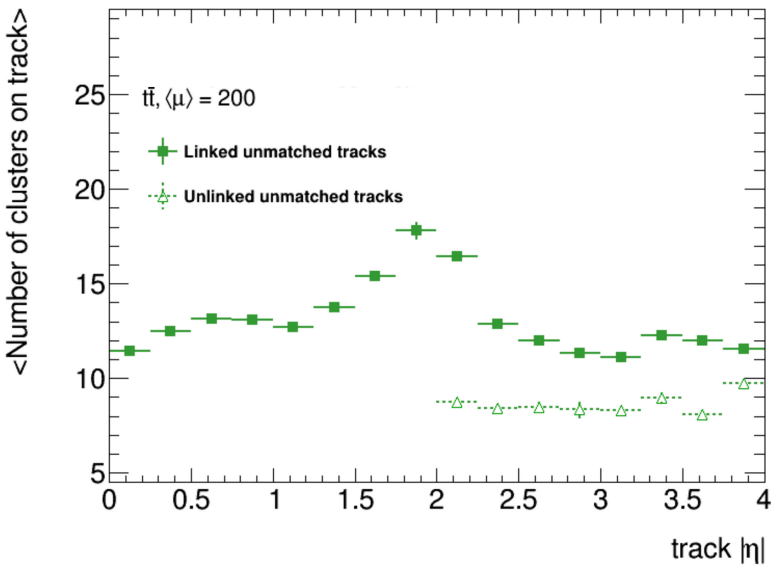


Figure 8.20: Trajectory size (number of hits) for linked and unlinked tracks.

in the figure, are quite far apart in the euclidean space but aligned along the angular distance and thus share many neighbors. Depending on the hits that are shared, overlapping buckets can produce similar seeds and later tracks. This results in duplicate tracks which is an issue we want to minimize.

Figure 8.22 shows the distribution of the overlap fraction between 10k randomly queried buckets. This distribution was averaged over 10 events. The wide majority of buckets do not overlap with any other bucket. The query of doublet hits however produces identical buckets (bin at 20) and has

to be prevented. A certain amount of overlap can be allowed in order to ensure a total coverage of the detector. This number is chosen as a trade-off between containing all the seeds and avoiding duplicate tracks (seeds).

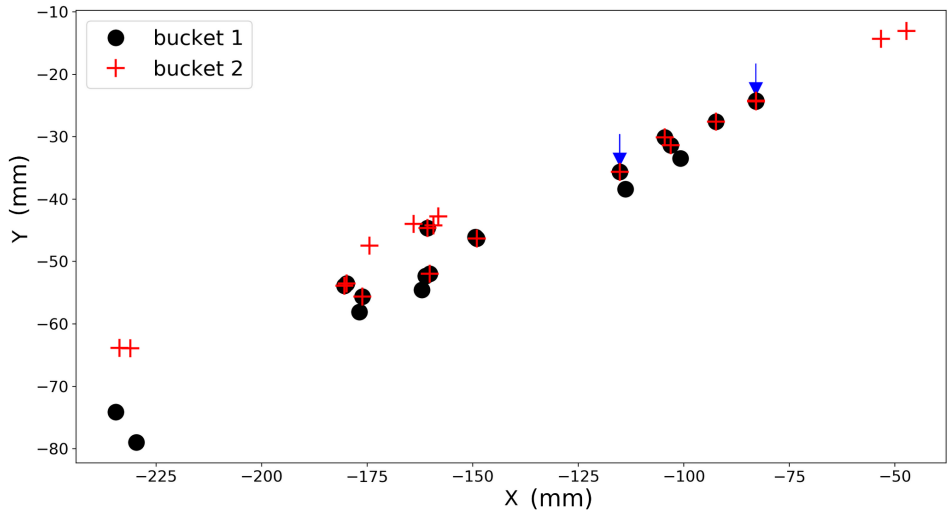


Figure 8.21: Example of two overlapping 20 hits ANN buckets. The query hits are marked with the blue arrows.

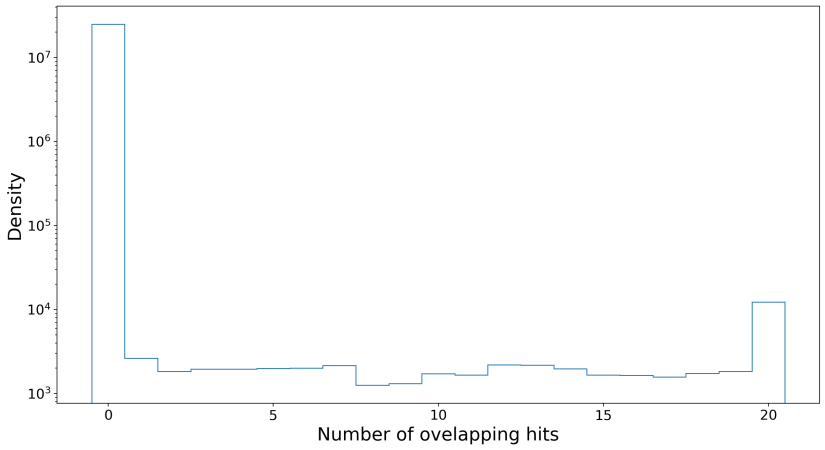


Figure 8.22: Example distribution of the number of overlapping hits between 10k ANN buckets of 20 hits.

The overlap removal is the process of discarding one bucket if a fraction of its hits are already contained in another bucket. Typically, this fraction can vary from 70% to 80%. Depending on the size of the buckets, if less than 70% of the hits are contained in two different buckets, then the non common hits (at least 30%) can be used to create different seeds. Therefore, buckets overlapping with less than 70% are not removed to ensure a high efficiency.

We test different overlap removal thresholds on one event and analyze their impact on the total number of seeds produced, the number of candidates and accepted candidates per bucket and finally the event efficiency loss. For these tests, we consider 33k ANN buckets of 70 hits. The different distributions are presented in Figure 8.23. In each case, the standard buckets with no overlap removal are shown in black and the alternative where buckets that contain

at least 70% of overlapping hits are discarded is shown in red. We can see that the number of buckets producing no seed has dropped by more than half. Although the largest impact is on empty buckets, the ones that produce seeds are also removed, i.e. decrease of the number of buckets producing seeds. This is directly reflected in the number of candidates and the number of final tracks (accepted candidates) shown in Figure 8.23(b) and 8.23(c).

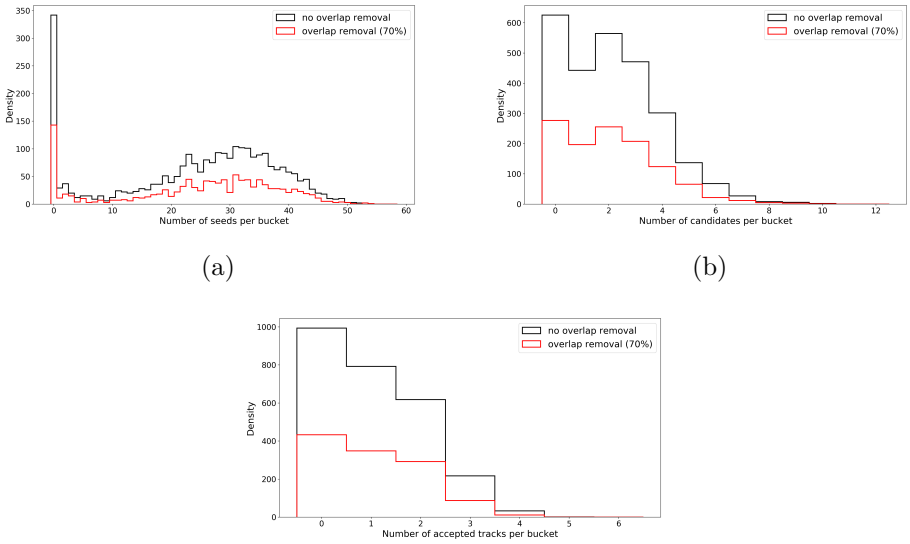


Figure 8.23: Impact of a 70% overlap removal on the number of seeds, candidate tracks and accepted tracks.

The overlap removal procedure checks the fraction of overlapping hits without any specific constraints on the position of common hits. Two buckets that have 70% of common hits might still produce different seeds if for example the remaining 30% unique hits are produced in the inner layers and satisfy seeding criteria such as a minimal distance between hits of at least 6mm. In this case, removing one of these buckets will result in the permanent loss of the seed (and later track) it contains. At the event level, this is demonstrated as a function of the pseudorapidity region in Figure 8.24. The number of discarded buckets in this example is :

- >85% overlap: 3K buckets were removed.
- >80% overlap: 5K buckets were removed.
- >70% overlap: 10K buckets were removed.

Firstly, the number of buckets removed at a threshold of 85% is quite high (3K). This number increases rapidly as the overlap threshold is lowered. Removing buckets that have at least 70% hits in common with other buckets decreases the total number of buckets considered by a third. This third of buckets contributed by an average of 6% to the event efficiency.

Secondly, it is interesting to note that as the overlap threshold is lowered, the event efficiency becomes highly dependant on the pseudorapidity region considered. For example, for $0.5 < |\eta| < 1.5$ and $3 < |\eta| < 4$ the efficiency is lightly impacted by the buckets removal. In $2 < |\eta| < 2.5$ however, the efficiency drops significantly compared to any other region. This is explained by the fact that the number of buckets generated per eta region is not exactly the same but rather a function of the material distribution in the detector

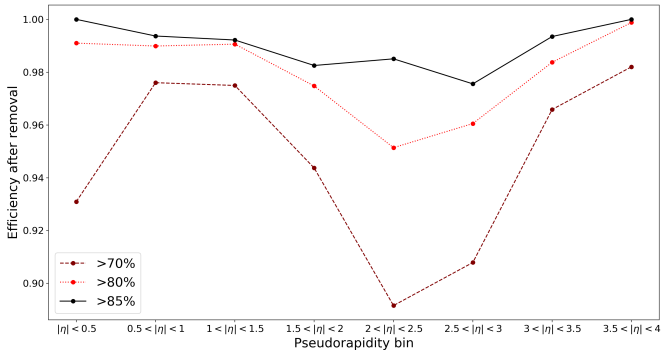


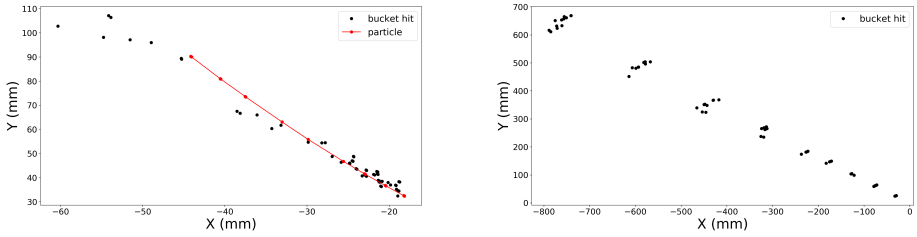
Figure 8.24: Impact of overlapping buckets removal. Different overlap threshold are considered : 70%, 80% and 85%. The resulting event efficiency after bucket removal is shown as a function of the pseudorapidity.

(true number of hits). A solution to this overlap removal dependence is to vary the overlap threshold in the different detector regions.

A more robust solution however is to automatically detect noise buckets and remove them completely additionally to removing highly overlapping buckets. Bucket filtering is the additional layer to detect empty buckets and automatically discard them.

8.8 Bucket filter

A bucket that contains a track has to contain 4 hits or more produced by the same particle. A bucket that does not satisfy this condition is a collection of random hits that do not contain any information. Such buckets (although a minority) have to be filtered before any reconstruction procedure is run. The filter extracts patterns that discriminate between good buckets and bad ones (noise buckets). Figure 8.25 shows an example of a good bucket with a leading particle of 14 hits in (a) and a noise bucket with no track larger than 3 hits in (b).

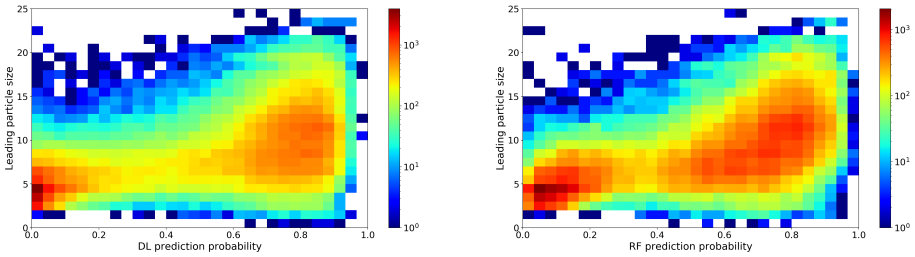


(a) (b)

Figure 8.25: Illustration of the filter task. (a) is a bucket that contains a track of 4 hits or more. (b) is a bucket with no track larger than 3 hits, i.e. noise bucket. The bucket filter task is to distinguish between such buckets.

The filtering of the buckets is in fact a binary classification : good (1) and bad (0). The classifier takes a bucket as input and produces a binary response. The bucket is a high dimensional object where each hit is a 6 dimensional vector: $x, y, z, \theta_{angle}, \phi_{angle}, pixel/strip$. The θ_{angle} and ϕ_{angle} are the inner angles of the hits cluster shape. The $pixel/strip$ is a binary value that takes 1 if the hit is a pixel and 0 if it is a strip.

In this study, we consider buckets of 50 hits. The classifier input is therefore a matrix of shape $(N, 50, 6)$ where N is the number of buckets considered. All input vectors are normalized to unity (between 0 and 1). Buckets are sampled from 10 events for training and we use 10 additional events for validation (prediction). The output of the model is set to 1 if a bucket contains at least 4 hits from the same particle and 0 otherwise. The training dataset is balanced to contain an equal amount of positive and negative buckets. Two different machine learning models are tested: a random forest (RF) and a neural network (NN). We use 200 estimators for the random forest. The NN has two fully connected hidden layers of 500 units each with batch normalization. A stochastic gradient descent is used to optimize the network weights and a binary cross-entropy loss penalizes the network error. Both models (RF and NN) output a prediction probability for each bucket. This probability is confronted to the leading particle size per bucket in Figure 8.26.



(a) (b)

Figure 8.26: Joint distribution of the leading particle size and the filter prediction probability. (a) shows the model response of the neural network while (b) shows the output probability of the random forest. Colors highlight the frequency of a pair (leading particle size - model probability).

A perfect model produces a linearly correlated response to the leading particle size. In both models we can see two distinct blobs around probability zero with leading particle sizes 4 and around higher probabilities (>80%) with a more spread leading particle size (centered at 10). With a probability cut of 50%, the majority of buckets returned have tracks of at least 4 hits. The error of the model, i.e. the fraction of false positives is 3%. The efficiency of the model, i.e. the fraction of true positives is of 82%. This means that around 18% of positive buckets were classified with a probability smaller than 50%.

Probability cut	30%	50%	70%
RF error (efficiency)	3.4%(82%)	3% (72%)	2.2% (44%)
NN error (efficiency)	3.4%(81%)	3.2% (72%)	3% (51%)

Table 8.3: Error and efficiency of the two models for different prediction probability cuts.

Table 8.3 shows the evolution of the models error and efficiency when varying the prediction probability threshold. We can see that both models show similar performances at different cuts. The random forest has a lower error at higher thresholds while the neural network has higher efficiency at higher thresholds. This behavior is also illustrated in the Figure 8.26 where the distribution of the random forest is more spread vertically around low probabilities, i.e. good buckets are associated to low prediction probability wrongly.

Concretely, the filter task is to shift the distribution of leading particle sizes towards higher values by filtering those with no track. The distribution of leading particle sizes before any filtering is shown in Figure 8.27 as the randomly selected buckets (black markers). The resulting distribution after applying a neural network (DL) filter with a probability cut of 50% is shown by the red markers in Figure 8.27. For reference, the true particle size distribution is highlighted in the background of the figure.

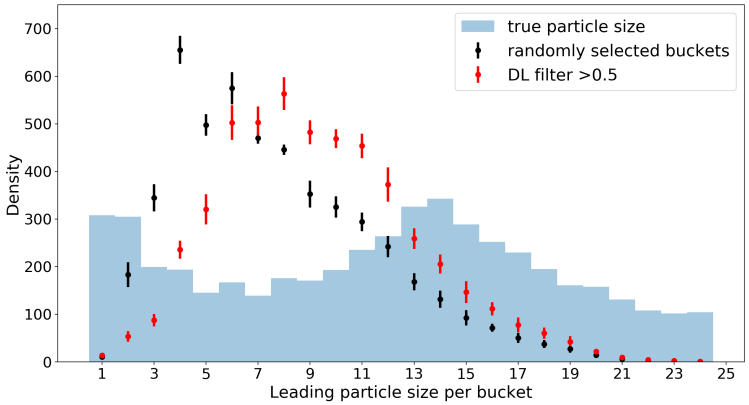


Figure 8.27: Impact of the bucket filtering on the leading particle size distribution.

The shift in distribution from the blue histograms to the black markers is attributed to the ANN buckets quality. The majority of buckets contain four, five and six hits tracks. The filter shifts the distribution further towards higher values. The number of buckets with non reconstructable tracks (<4 hits) is significantly reduced. After the filtering and on average, a bucket contains 8 hits produced by the same particle. The distributions shown is Figure 8.27 were obtained by averaging the model response on 10 unseen events.

Adding more input information to the model helps in increasing the overall accuracy. The cluster shape information, for example, is a valuable feature for extracting the descriptive patterns of a bucket. In Chapter 10 we present a pipeline model that take as input the full information captured by the detector: raw hit features, activated clusters as images as well as the detector geometry information.

8.9 Summary and conclusions

This chapter presented a full evaluation of the fast similarity approach on the ITk simulation dataset. The proposed approach being first designed and tested on the TrackML challenge, it is expected that its application on a realistic simulation dataset presents many more challenges. However, despite the enumerated dataset differences and the need for adjusting ANN parameters such as the total number of queries, the fast search approach allowed to define fast reconstructable buckets. Due to the noise level in the ITk dataset, the number of necessary ANN queries grew from 4K (on the TrackML dataset) to approximately 70K while the size of the bucket was increased to 70 hits (instead of 50 hits). Moreover, a necessary addition was the introduction of displaced buckets that capture tracks with a z offset.

Additionally, this chapter established the possibility of running the ATLAS standard tracking chain on a very small number of hits as opposed to

the full event. While the standard seeding and track finding algorithms were designed to process thousands of points in input, they successfully constructed tracks in buckets of 50 and 70 hits. In order to cope with the noise level in the ITk dataset, buckets were tested for seeding only since the ANN buckets were unable to contain the full particle trajectory. However, this new direction showed promising efficiency and timing results compared to the standard tracking in the full event. Tracking in buckets allowed to retrieve more tracks in the low P_T region.

The two shortcomings of the presented approach, namely duplicate tracks and noise buckets, were addressed with an overlap analysis and a filtering model. The approach presented in this chapter can show an important speed-up potential when the ongoing multi-threading and GPU tracking algorithms are fully deployed. The parallel tracking can be performed in buckets on thousands of threads.

While the fast search approach consistently contained tracks and seeds, the metric learning model presented in Chapter 7 was not able to cope with the ITk dataset challenges. As a consequence, in Chapter 9, we propose to design a novel metric learning model that explicitly addresses the characteristics of the ITk dataset.

Bibliography

[1] Hashing for track reconstruction : Hashing and similarity learning for track reconstruction https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/IDTR-2019-008/

9

TrackNet : Tracking aware embeddings

Standard metric learning techniques yield poor performances on the ITk simulation sample. In this chapter, we propose a novel model that performs *tracking aware* metric learning.

9.1 Motivation

Chapter 7 discussed the successful application of LFDA and UMAP on the TrackML dataset. Both techniques, using supervised particle information, were able to produce mappings that clustered together hits produced by the same particle. As discussed in previous chapters, these mappings were possible due to the properties of the TrackML dataset (low noise level). The classes defined by the particle label information allowed to learn consistent patterns and shapes that accurately generalized to unseen events. When training an UMAP model on the ITk simulation sample, the resulting 2D mapping does not reassemble at all the desired output[1]. Figure 9.17(a) shows the resulting mapping of the hits in the central pseudorapidity region $|\eta| < 1$ as a grey background as well as a selection of 10 long particles (with at least 12 hits). Hits produced by the same particle have the same color and are linked by continuous lines.

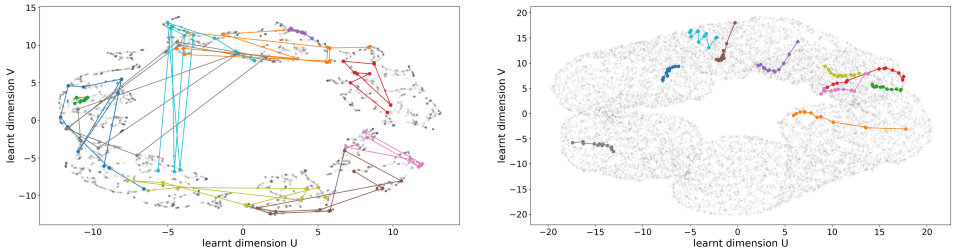
The large spread of similarly colored points in Figure 9.17(a) is an indicator of a poor mapping. As a comparison, a similar subset (similar pseudorapidity region and number/size of highlighted particles) from the TrackML dataset is mapped with UMAP in Figure 9.17(b). We can see that although the mapping is not perfect, different particles are well separated and all the hits are mapped into different locations as opposed to the seemingly random behavior seen in Figure 9.17(a).

UMAP is considered the current state of the art in manifold learning yet it fails to capture track patterns in the ITk simulation sample. At this stage, this inability to generalize from TrackML to ITk is expected due to the numerous differences. We use the lessons learned from previous comparisons to propose a custom metric learning model.

9.2 The TrackNet loss function

In deep learning models, the loss function establishes the desired output and penalizes the network accordingly. The purpose of the model is to estimate

[1] This is despite running a grid search on all the UMAP parameters



(a) (b)

Figure 9.1: Mapping of particles using UMAP applied to (a) an ITk event and (b) a TrackML event.

from a dataset X some target values $\hat{y}$ that approximate as much as possible the true target values y. The loss function quantifies how close is the model prediction to the true values. It is therefore of the form :

$$\mathcal{L} = \gamma(\hat{y}, y) \tag{9.1}$$

where γ is a function that measures the agreement between predictions and true values. For example, one of the most popular loss function is the Mean Squared Error (MSE) where $\gamma = \frac{1}{n}\sum_{i=0}^{n}(\hat{y}_i - y_i)^2$ and n the total number of data points.

Many forms of the loss function exist. However, these standard functions apply well to standard problems (classification, estimation, regression). Our goal with the metric learning model is to assign new coordinates to every hit such that tracking becomes intuitive. Characteristics of this intuitive output have to be encoded in the loss function.

Figure 9.2 shows the global model architecture as well as the input and output. The model takes in hits and returns them in a new d dimensional space. A bucket of n hits can be given as input to the model. To illustrate the model actions, Figure 9.2 shows two particles as input : P1 and P2. Each hit in the input is an m dimensional feature vector. We use all the available information per hit : x, y, z coordinates, layer and volume identifiers, pixel inner angles and the modules directions. In total 14 variables per hit. The contributions of the input features is also discussed in this chapter.

The different layers of the neural network propagate the hits using a $\tanh$[2] function on the learned weights to produce the output hits (2 dimensional in the figure). The actions of the loss function are illustrated in the figure through the arrows directions. An intuitive tracking space is one where hits from different particles exist in different locations and those produced by the same particle are close-by. The loss function tries then to pull apart hits from different particles (action of the outward large arrows) and pushes together same-particle hits (action of the inward small arrows).

In Figure 9.2, the output of the model constitute two distinct blobs where one contains hits from P1 and the second hits from P2. Additionally to the pushing and pulling actions, a clustering feedback computed on the output is incorporated into the loss value at the end of every epoch.

The proposed loss form is introduced in Equation 9.2. It has three weighted components : The compactness term $\mathcal{L}_c$, the isolation term $\mathcal{L}_I$ and the clustering term $\mathcal{L}_{CL}$. The compactness and isolation enforce the push and pull

[2]Chosen after a grid search. More details can be found in the Model fine tuning section 9.3

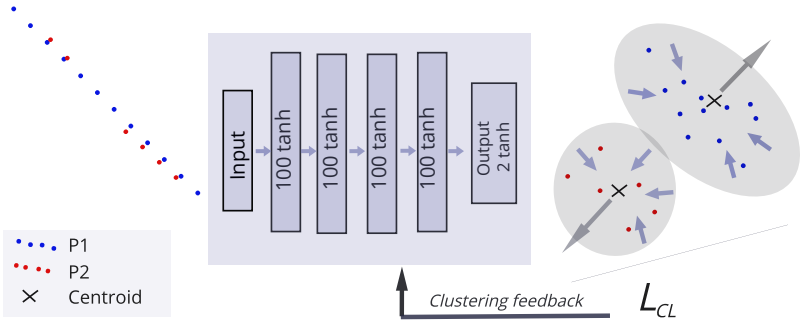


Figure 9.2: The TrackNet model. Learns to map input hits into a new feature space where particles are well separated.

effects while the clustering term provides a general feedback on quality of the clusters in each epoch.

$$\mathcal{L}_{TrackNet} = (\alpha\mathcal{L}_C + \beta\mathcal{L}_I + \gamma\mathcal{L}_{CL})^{\zeta} \tag{9.2}$$

$$\mathcal{L}_C = \sum_{i=0}^{K} S(c_i) \quad ; \quad \mathcal{L}_I = \frac{1}{S(\mu_{[1..k]})} \quad ; \tag{9.3}$$

The impact of each term of the loss function is regulated through the weights : α, β, γ. The ζ power of the loss takes values starting from 2 to ensure a larger penalty for larger errors. The compactness loss, detailed in Equation 9.3, addresses the variance $S()$ within each particle cluster c_i. It is computed for every K clusters in the batch. The variance is defined as $S(x) = \frac{\sum_{i=0}^{n}(x_i - \overline{x})^2}{n-1}$ where x is the ensemble of points of interest. The smaller the variance of a particle in the learned space, the smaller the model penalty. The isolation, expressed in $\mathcal{L}_I$, concretely isolates the centroids μ of the particles. The network learns to shift the hits of a particle cluster such as their centroid is sufficient far from other centroids. In turn, the compactness term ensures that the hits within a particle are close enough that their centroid remains a good representative. As a consequence, the higher the variance of the particle cluster centroids, the larger the isolation. In $\mathcal{L}_I$, we choose to minimize the inverse of centroids variation.

Although the compactness and isolation terms seem to describe the desired output space, there are cases where the two losses are minimized but the output space is not ideal. For example, the majority of particles can be pushed apart into different locations but a hit or two can *escape* from this location. This is because both the variance and the mean operate on the average value and are therefore *robust* to outliers, i.e. a wrongly mapped hit does not heavily impact the mean and variance. Moreover, since the goal of the TrackNet model is to facilitate the use of a clustering technique to retrieve the tracks, a clustering quality metric is a powerful addition to the loss function.

We chose as a clustering metric the silhouette coefficient SC (introduced in Section 7.2.4). This metric is defined for every hit in the output space and the returned value is the average over all the hits. The clustering loss $\mathcal{L}_{CL}$ is exactly this metric averaged over all mapped hits in a given batch. The silhouette coefficient loss term is defined as:

$$\mathcal{L}_{CL} = \frac{b - a}{\max(a, b)} \tag{9.4}$$

With a the mean distance between a hit and other mapped hits produced by the same particle and b the mean distance between a hit and all other hits in the next nearest particle cluster. The strength of the silhouette coefficient is that it considers all distances between hits. It varies between -1 and 1 where 1 describes perfect clusters (well separated). Towards 0, the clusters are overlapping and negative values indicates random clusters. The SC range of variation is used throughout this chapter to compare different model configurations.

An additional evaluation metric

Although the silhouette coefficient sufficiently describes the quality of a clustering, we might be interested in a fraction of the hits only. For example, if a bucket contains 20 hits, of which 10 belong to the same particle and the remaining points represent noise hits (2 or 3 hits per particle), we are interested in the clustering of the leading particle (10 hits in this case) and not in the shorter tracks. We propose to add an additional evaluation metric that focuses on the mapping of the leading particle in the model output. Such metric combines both the efficiency and purity of the cluster of interest c and is known as the Intersection Over Union (IoU) or the Jaccard Index. Effectively, it is the intersection of prediction and truth over their union:

$$IoU_c = \frac{\sum_{n=0}^{N} \mathbf{1}[y_n = c \ \& \ \hat{y}_n = c]}{\sum_{n=0}^{N} \mathbf{1}[y_n = c \text{ or } \hat{y}_n = c]} \tag{9.5}$$

where $\mathbf{1}$ represents an indicator function that takes a value of 1 if the condition is met and 0 otherwise. In the standard definition of IoU (generally used for classification tasks), y_n is the truth label of the data point n and $\hat{y}_n$ the predicted label. c is the class of interest, i.e the leading particle cluster. We propose the following formula:

$$IoU_{particle} = \frac{\sum_{n=0}^{N} [y_n = p \ \& \ c_n = \hat{p}]}{\sum_{n=0}^{N} [y_n = p \text{ or } c_n = \hat{p}]} \tag{9.6}$$

where y_n is the true label of the hit n and c_n the corresponding clustering label. p is the identifier value of the leading particle and $\hat{p}$ the majority clustering label in the leading particle. Figure 9.3 shows an example of a clustering label assignment (numbers) and the truth hit identifiers (colors). In the example, the leading particle is the one shown in blue ($p = blue$) and the majority corresponding clustering label is $\hat{p} = 250$. In this case, the IoU is equal to $\frac{6[blue \textbf{ and } 250]}{7[blue \textbf{ or } 250]} = 0.85$. The silhouette coefficient value in this example is much smaller (approximately 0.6) since the blue point wrongly labelled 10 is much closer to its true neighbors (blue) than similarly labelled points (red).

A clustering algorithm

The computation of the IoU metric requires a labelling of the hits, i.e. a clustering algorithm. Applying a clustering algorithm on the output of the TrackNet model allows to retrieve particle tracks and therefore to evaluate the quality of the model. The complexity of the selected clustering algorithm is inversely proportional to the quality of the TrackNet mapping. Indeed, the majority of existing clustering algorithms would successfully retrieve particle traces if *all* learned mappings fulfill the requirements of *isolation* and *compactness*. Section 9.4 describes a novel clustering strategy proposed and designed to run on the output of the TrackNet model. The optimization of the TrackNet model however is independent of the clustering technique used. As a consequence, the optimization of the model is conducted with a simple Agglomerative Clustering (AC) with a variable threshold distance. Details on

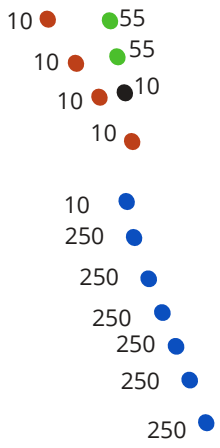


Figure 9.3: Illustration of a clustering output. The labels predicted by the clustering are encoded as numbers and the truth hit identifiers as colors. The leading particle is the blue one with 7 hits, 6 of which are correctly assigned similar labels by the clustering algorithm.

the standard AC algorithm are presented in Chapter 4. The threshold distance is the maximum distance at which the algorithm stops merging clusters. A dynamic distance means that the most suitable distance value will be selected depending on the TrackNet mapping (with a grid search), i.e. it is considered as an additional parameter to tune. In the remainder of this chapter, all IoU values are computed using an agglomerative clustering.

9.3 Model Fine Tuning

Many input parameters play an important role in the mapping of hits from their raw feature space into a new meaningful coordinate system. The fine tuning of these parameters as well as the understanding of their impact is crucial. Very often, a slight variation in any of the parameters can result in large disruption of the output. The TrackNet model relies on:

- The input shape : The dimension of the hits to map including global coordinates of the hits, layers and volumes, cluster angles and modules orientations. We evaluate different combinations as well as considering all of the available features.
- The model architecture : Number of layers, number of units per layer, activation functions, optimizer, batch size, number of epochs.
- The input hit pseudorapidity. We evaluate the model on all the detector as well as on individual pseudorapidity bins.
- The loss function weights: Studying the effect of putting more weight into the compactness or into the isolation.
- The dimensionality of the output space: 2D mappings allow to visually inspect and understand the model output whereas higher dimensions generally allow a better separation of the particles.

In the remainder of this section all the enumerated parameters are discussed and evaluated one by one. The number of hits passed to the model is selected

depending on the application. Along this chapter, we choose a bucket size of 20 hits as input to the model. This is motivated by earlier studies that showed a 20 hits bucket has a high enough probability to capture a track while maintaining a small size for a faster clustering.

9.3.1 The model input

Every hit in the dataset is described by the following features:

- x, y, z : global coordinates.
- Layout coordinates: barrel endcap index, layer disk index, eta and phi modules.
- List of activated pixels: (eta coordinate,phi coordinate,ToT).
- Inner angles of the activated pixels: eta angle, phi angle.
- Direction of the module surface in the detector, i.e. norms along x, y and z.

Similarly to the study conducted in Chapter 8, the TrackNet model is trained on pixels only. All of the above features are considered except for the list of activated pixels[3] (the inner angles are however considered). The model architecture search cannot be separated from the input shape analysis. An architecture might work for a purely 3 dimensional input while a different architecture is needed if more features are considered. The grid search is therefore ran on the input shape and the model architecture simultaneously thus exploring all possible combinations.

Figure 9.4 shows the result of a grid search on the model architecture and the input features along the two evaluation axes: SC and IoU. Each dot (or cross) is a variation of either the model architecture parameters (or input features). The number of epochs for the training of the model is color coded in the figure.

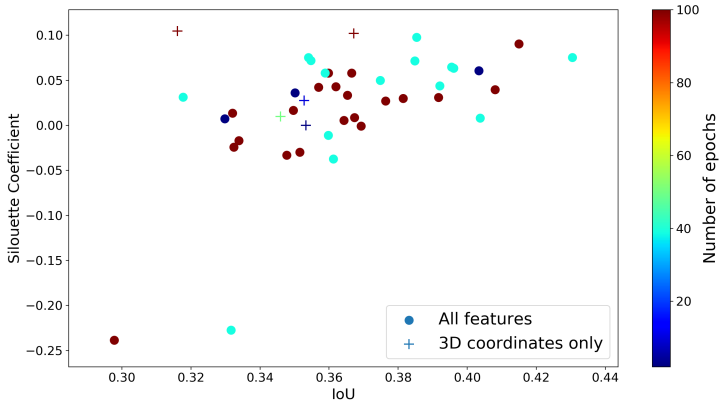


Figure 9.4: Model performance with different architectures and input features. Crosses denote a 3D input while the dots show the performance when considering all the available variables as input. Colors indicate different number of epochs.

[3]Including variable size images as input is not feasible with the selected network architecture. In Chapter 10, we propose a model that takes these images as input.

The best performing configuration is a 4 layer fully connected network where each layer is composed of 200 units and a *tanh* activation function. We use a batch size of 500. As can be seen by the colors spread, a higher number of epochs does not necessarily improve the performances. The ideal number of epochs is determined through early stopping. Early stopping is a binary feature in neural network training. If it is turned on, the model will monitor the evolution of one interesting variable (usually the validation loss) and automatically stop the training if its values do not improve over a certain number of epochs (referred as *tolerance*, usually 5 or 10 epochs). In our application, we choose to monitor the validation loss with a tolerance threshold of 10 epochs.

Feature importance

In order to evaluate the importance of the hit features, a common approach is to alternatively set a feature value to zero and analyze the impact on the output representation. By doing so, features with low contribution (lower network weights) will minimally affect the mapping whereas the absence of important features will significantly worsen the result. To quantify the *impact* in a metric learning application, we propose to measure the shift in the particle cluster variance and centroid coordinate of the leading particle. Our reference space is a mapping that considers all the input features. At every step, we select one feature from the input and set all its values to zero. In the resulting mapping, we compare the new position of the leading particle with the reference one. Figure 9.5 illustrates this comparison on two example variables.

The input hits (right most plot) are shown in the longitudinal plane and the leading particle is highlighted in red (4 hits). In the central plot and as an example, the norm of the module along the x axis is set to zero while in the left plot the phi coordinate of the module is neglected. For comparison, the reference mapping of the leading particle with all the features is shown as a grey discontinuous line. In this example, ignoring the phi module of the hit changes the output mapping noticeably more than ignoring the norm along x and the resulting mapping is improved (more compact cluster).

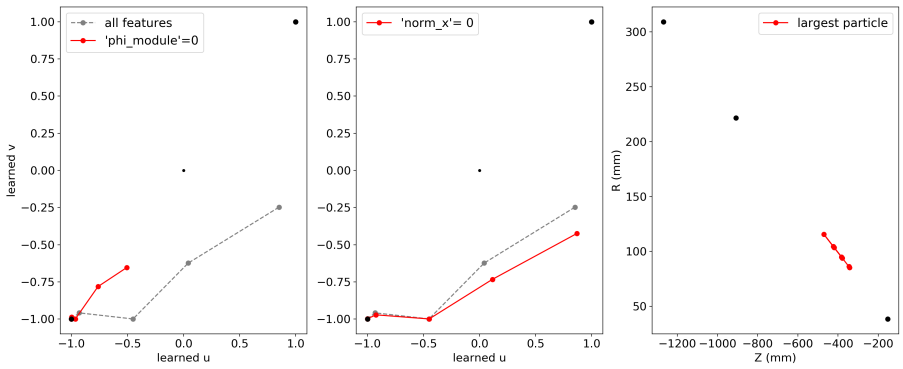


Figure 9.5: Illustration of the feature importance evaluation strategy. The leading particle is highlighted in red.

We quantify then the impact of the different features by measuring the shift in the mapping of the leading particle. In the example above, the shift is between the reference mapping (grey discontinuous line) and the mapping with a feature set to zero (red continuous line). The change in the mapping is evaluated by measuring the euclidean distance between the respective

centroids and the change in the cluster variance. The further away the new centroid of the particle is from the reference position, the higher the impact (positive or negative) of the feature. The change in variance however can be interpreted directly as negative if it increases and positive if it is smaller as the goal remains to map the particles into compact clusters. As a result, in the example mapping shown in Figure 9.5, removing the phi_module information from the input improves the mapping.

The results of the different shifts in the mean and the variance of the mappings are summarized in Figure 9.6. These quantities are averaged on an unseen event.

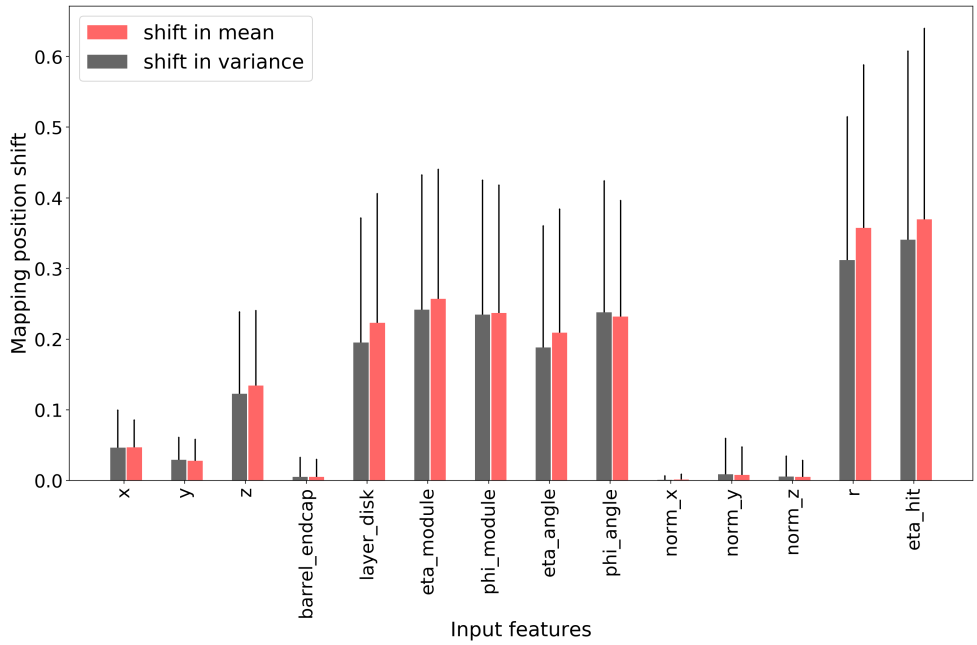


Figure 9.6: Input features impact on the mapping of the leading particle.

It is interesting to note that the geometry related features (layer disk, eta and phi modules coordinates) play an important role in the model mapping. As expected, the r and eta of the hit have the highest impact on the model output. The inner angles of the hit clusters contribute significantly to the output despite their lower[4] resolution. The analysis of feature importance and subsequently the selection of the best features allows to increase the interpretability of the model and reduce significantly the training time. Figure 9.5 demonstrates that the module norm information along the three axis as well as the barrel_endcap coordinate do not impact the mapping and can therefore be removed.

In the next section, we study the case of training different models in different pseudorapidity regions.

9.3.2 TrackNet and Pseudorapidity

In the previous sections, the TrackNet model is trained on the full detector. It was observed throughout this work that ML models yield different performances depending on the pseudorapidity region considered. In chapter 7 and 8 different performance evaluations showed as slight drop in efficiency in the central region compared to the forward region. The TrackNet model is no exception to this behavior since the same TrackNet configuration, evaluated

[4] As compared to the TrackML dataset.

in different pseudorapidity regions results in widely different scores. Figure 9.7 shows the different results obtained in 8 pseudorapidity bins each with a 0.5 width. The evaluation axes are the average IoU and the average SC for all possible buckets of 20 hits on an unseen event. The pseudorapidity range is annotated on top of the markers.

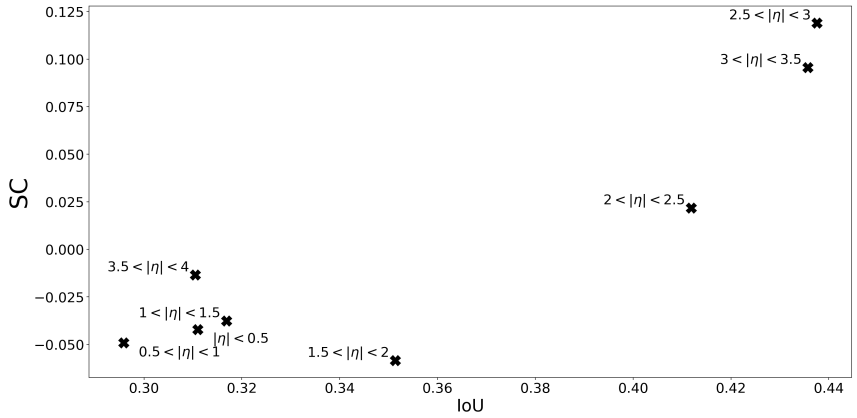


Figure 9.7: Spread of the model performance in different pseudorapidity region.

Since the same model configuration works differently depending on the pseudorapidity bin considered, we propose to optimize and train a TrackNet model per bin.

TrackNet performance in $|\eta| < 1$

Figure 9.8 shows the performance of the model when considering only the central pseudorapidity region $|\eta| < 1$. The performance is presented by the joint distribution of the SC per bucket and the IoU per leading particle. We also highlight the coordinates where charged high P_T particles are strongly present (>10%) by red squares. We choose to maintain the full SC range in the figure to highlight the gradual shift from pseudorapidity region to the next, i.e. more forward region have higher SC values. As a reminder, SC values close to 1 denote a good clustering with respect to all the hits present in the bucket. Values close to 0 denote overlapping clusters.

As expected, the separation of the leading particle is better than for the rest of particles. This is apparent from the higher values of the IoU compared to the SC. This is also the case for all the pseudorapidity regions.

TrackNet performance in $1 < |\eta| < 2$

As expected, the performance in the following pseudorapidity region improves significantly. This is despite optimizing each model individually and selecting the best configuration *per bin*.

Figure 9.9 shows the performance of the mode in $1 < |\eta| < 2$ region. This pseudorapidity region shows higher SC and IoU values. Charged, high P_T particles are mostly located in buckets with a 50% SC score and a 60% IoU score.

TrackNet performance in $2 < |\eta| < 3$

Starting from $2 < |\eta| < 3$, the correlation between SC and IoU becomes significant in the performances plot on Figure 9.10. Many more buckets have high SC scores (>30%) and high IoU scores (>60%). The TrackNet model

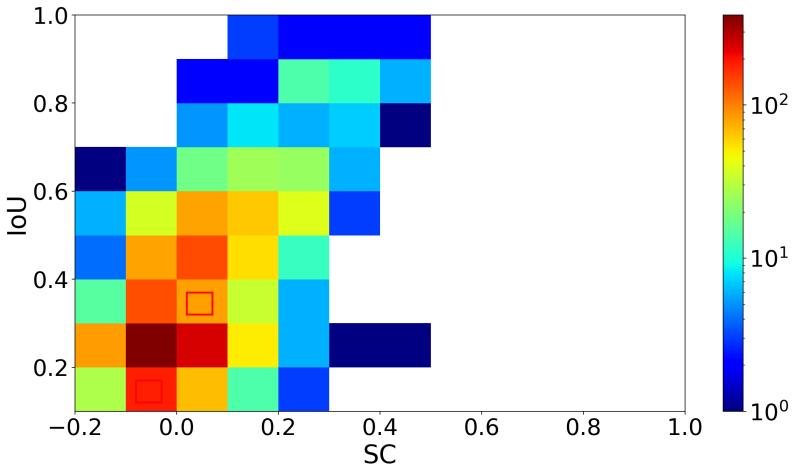


Figure 9.8: TrackNet performance in the central η region ($|\eta| < 1$). The red square represents charged particles with a P_T greater than 900MeV.

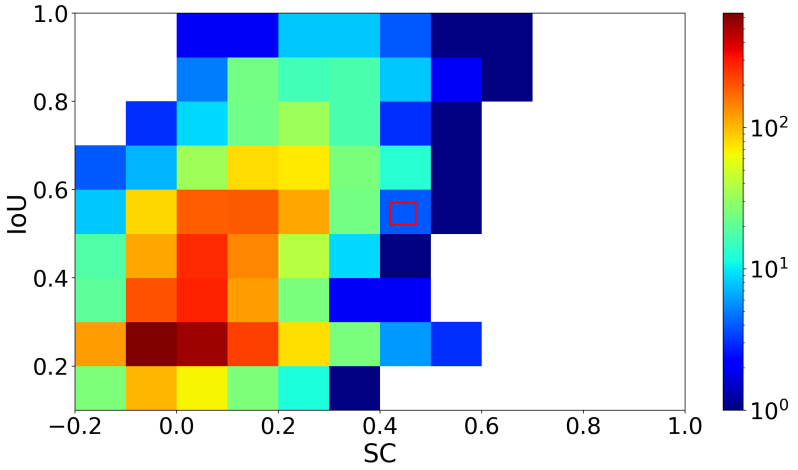


Figure 9.9: TrackNet performance in the $1 < |\eta| < 2$ region. The red square highlights bins with an important fraction (>10%[5]) of charged particles and P_T>900MeV.

exhibits the best performance in this region with an average SC of 13% and an average IoU of 46%.

TrackNet performance in $3 < |\eta|$

The results from the best performing TrackNet model for the forward region ($3 < |\eta|$) are summarized in Figure 9.11. Charged, high P_T tracks have noticeably lower IoU values in this region. This is consistent with a similar drop in efficiency, also at the edge of the pseudorapidity coverage, presented in Chapter 2.2.3.

The red squares in the different performance plots do not exhibit a consistent pattern, i.e. charged, high P_T tracks are not necessarily easier to retrieve.

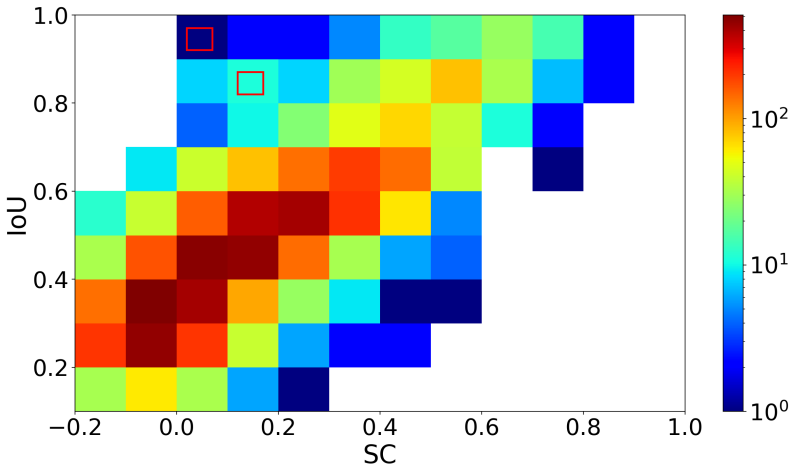


Figure 9.10: TrackNet performance in the $2 < |\eta| < 3$ region. Red squares highlight charged particles with P_T>900MeV.

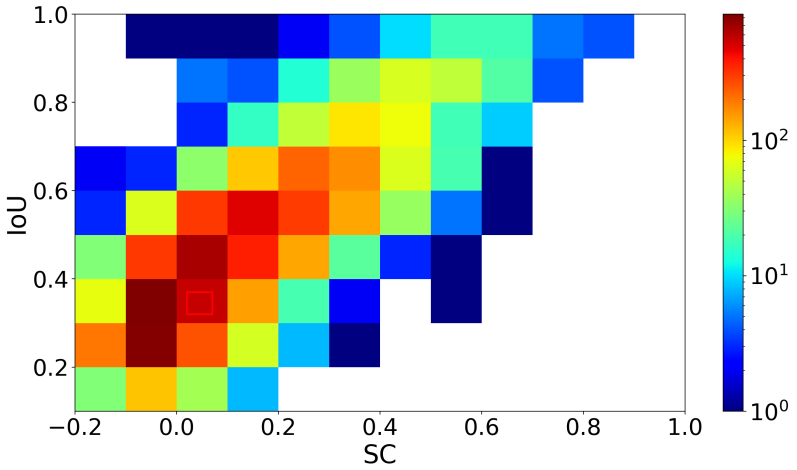


Figure 9.11: TrackNet performance in the $3 < |\eta|$ region. The red square highlights charged particles with P_T>900MeV.

9.3.3 Output Dimensions

In an application where the goal is to map hits into a new feature space, the number of dimensions the model is allowed to use is crucial. Intuitively, the higher this number the easier it is to disentangle hits. In this section we analyze the mapping performances when increasing the number of output dimensions from 2 to 20. The maximum output dimension size is selected to be 20 since the available number of input feature is 14 and a higher dimensionality is not necessary. Figure 9.12 shows the performances of different output dimensionality choices. Each marker represents the best scoring model configuration for the current dimension. This means that not only the size of the output is different but also the model architecture and convergence. For example, the majority of tested models[6] converged after approximately 25 epochs but some needed 100 epochs. It is worth noting that an increase in the dimensionality of the output space does not necessarily improve the mapping. Moreover, the

[6]The *best* performing models are shown in Figure 9.12 and Table 9.1. Approximately ten models have been trained and tested per output dimension.

Output dimension	Epochs	Learning rate	Number Nodes	Avg SC	Avg IoU
2	92	10^{-3}	50	0.07	0.44
3	100	10^{-3}	50	0.11	0.44
4	21	10^{-2}	200	0.09	0.46
5	84	10^{-3}	50	0.09	0.45
6	100	10^{-3}	50	**0.10**	**0.46**
7	100	10^{-3}	50	0.08	0.44
8	100	10^{-3}	50	0.08	0.47
9	14	10^{-3}	200	0.09	0.43
10	31	10^{-3}	500	0.08	0.43
20	11	10^{-3}	50	0.08	0.42

Table 9.1: Configuration of the best performing models per output dimension.

performances variation along the SC axis is more prominent. This is explained by the higher sensitivity of the SC metric where a bad mapping of few hits has an impact on the value.

The TrackNet mapping that maximizes **both** metrics has 6 dimensions.

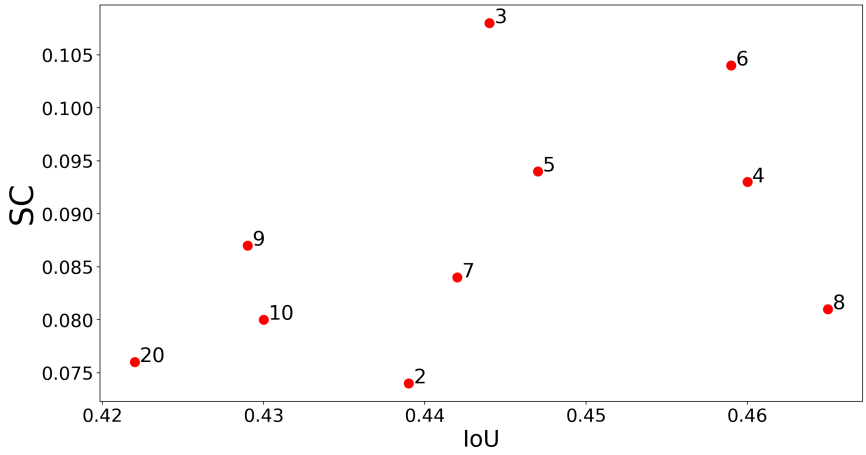


Figure 9.12: Performance evaluation of different output dimensionality. Annotated text indicate the dimension of the model output space.

The configurations of the best performing models per output dimension are summarized in Table 9.1. Half the models reach their optimal performance after 20 epochs. Interestingly, the models that require more epochs also require less hidden units: Only 50 hidden units per layer. This is 4 to 10 times less than the size of the models that converge after fewer epochs. The only exception to this behavior is the model that maps a bucket into a 20 dimensional space. The remaining hyper-parameters, activation function and batch size, are similar for all tested models. The activation function for the hidden layers as well as for the output node is a tanh. The batch size consists of 500 buckets.

9.4 Dynamic Tracking Linkage : A new clustering approach

The clustering algorithm that takes a TrackNet mapping to produce particle tracks cannot solely rely on the mapping quality. Hits that are merged into

clusters have to make sense from a physics (or physical) perspective. In this context, we propose a novel clustering algorithm that incorporates **tracking** knowledge and automatically predicts the stopping (merging) criteria, making it **dynamic** : Dynamic Tracking Linkage (DTL).

An example input-output of DTL is illustrated in Figure 9.13. The input of the clustering algorithm is a collection of hits mapped to a new feature space (6 dimensional as determined in the previous section). The output is a list of tracks. Contrary to standard clustering algorithms, DTL does not necessary label every hit in the input. The example shown in Figure 9.13 illustrates this characteristic. From the 20 hits present in the input, only 13 are labelled (8 and 5 hit tracks). The clustering of "track 1" is a good example of why a tracking oriented clustering is needed. While the first hits in track 1 form the closest neighbors (highest u,v coordinates), the last 3 hits are *contaminated* by noise hits (marked as crosses). The DTL algorithm, using tracking knowledge and dynamic stopping criteria ignores noise hits despite the fulfilment of the distance criteria.

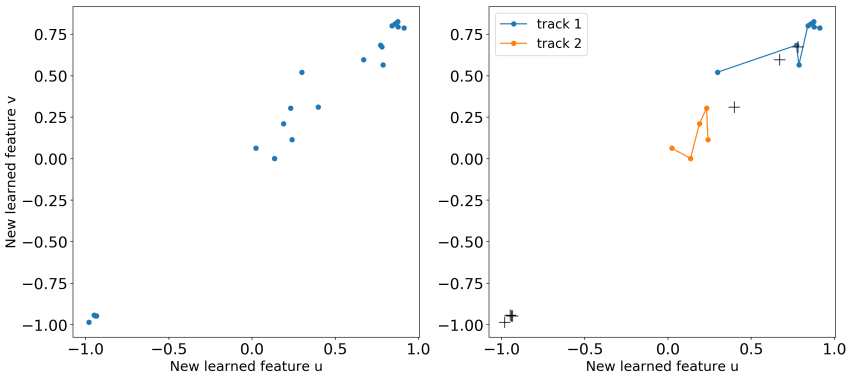


Figure 9.13: Clustering hits into tracks. Closest neighbors do not necessarily belong to the same track.

The specifications (and novelty) of the DTL algorithm are:

- The algorithm starts by simultaneously forming kernels of hits using the smallest euclidean distances.
- Only kernels of at least 3 hits are considered.
- Neighboring hits that fulfil tracking constraints are merged into clusters.
- Consistency of the clusters is determined automatically by a trained binary classifier.
- Consistent clusters are returned as tracks.

Figure 9.14 describes that DTL clustering strategy. In the remainder of this section, we first enumerate the pairwise tracking constraints, then the "consistency" of a cluster is defined through a classifier. Finally, we evaluate the TrackNet+DTL tracking pipeline.

9.4.1 Pairwise tracking penalty

The tracking constraints penalize an association of two hits that violates detector geometry properties. Hits that belong to different sub-detectors or similar ones but different regions are not considered by this penalty. The following

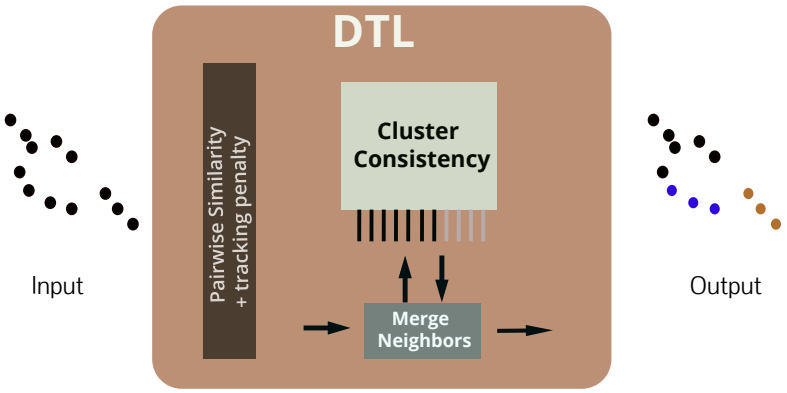


Figure 9.14: The DTL clustering strategy.

conditions denote the cases where a pair of hits is penalized and excluded from forming a cluster.

- If pixel hits occur on the same layer and the same module (in the eta or phi coordinates).
- If non endcap strip hits occur on the same module (in the eta or phi coordinates) with a distance in r smaller than 1mm.
- If endcap strip hits occur on the same module (in the eta or phi coordinates) with a distance in z smaller than 1mm.

The penalty is implemented as a large weight added to the euclidean distance of the pair. The DTL similarity distance between two mapped hits $h1$ and $h2$ becomes:

$$Distance(h1, h2) = Euclidean(h1 + h2) + Penalty(h1, h2) \tag{9.7}$$

Where Penalty() is a function that returns a large weight ($>> 1$) if any of the previously enumerated conditions are met and 0 otherwise.

To illustrate the impact of the penalty function in the new distance computation, we assign different weights depending on which of the previously listed conditions is met. For example, if the first condition (pixel hits on the same layer and module) is met, the penalty has a fixed of value of 7. The second condition is associated with a weight of 8 and the third with a weight of 9. The weight values can be chosen arbitrary as long as they are greater than the maximum distance[7] allowed by the mapping. Figure 9.15 shows the distribution of the DTL pairwise distance for **negative** pairs as compared to a standard clustering using only the euclidean distance.

Samples with coordinates 7,8 and 9 are negative pairs that would have had a 0 distance based on the TrackNet mapping. Generally, all the samples with distances ≥ 7 are excluded early on by the DTL distance function resulting in less false positives.

After defining the distance computation space and the tracking exclusion rules, the next step is the formation of the clusters. The merging of hits into clusters is referred to as *linkage*. The DTL algorithm uses a trained classifier,

[7]The tanh function bounds values between -1 and 1. The maximum distance is the one between the coordinates [-1,-1] and [1,1] : $\sqrt{8}$.

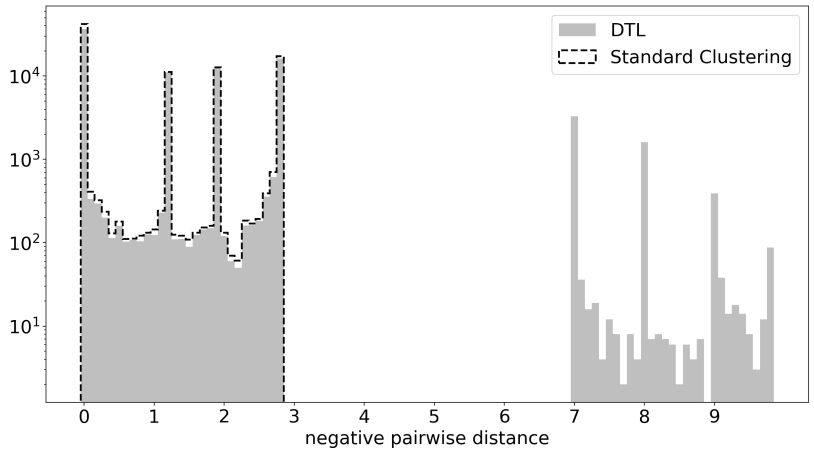


Figure 9.15: Distribution of negative pairwise distances in DTL as compared to a standard clustering algorithm (euclidean distance).

referred to as cluster consistency model, to determine whether clusters should be merged or not.

9.4.2 Cluster Consistency

We propose to build a classifier that decides if a hit should be merged with the current cluster. More specifically, the classifier will produce a probability on a set of hits at every stage of the clustering algorithm. The consistent clusters (final tracks) correspond to the latest clusters prior to the change from a positive to a negative prediction.

Throughout this work, multiple variants of hit classifiers have been described. Some models relied on a 3D hit input while others used cluster images (also combining both). All of the models aimed at demonstrating the feasibility of deciding whether a set of hits were produced by the same particle or not. The cluster consistency constraint also aims at deciding if a group of hits were produced by the same particle. The major difference to any of the presented models (Bucket filter in Chapter 8 and later VCS-Conv in Chapter 10) is the variability of the input size. Indeed, the cluster consistency model can produce a decision on an input formed by 4 hits, 5 hits...up to 10 hits. While 4 hits represent the minimal requirement to form a seed, 10 hits represent a good track estimate (on average, a particle is formed by 10 hits). The size of the model input has therefore 10 units. When fewer hits are passed to the model, the remaining units are deactivated, i.e. their values are set to zeros. The training dataset contains variable lists of hits along with a binary indicator that takes 1 if the hits are produced by the same particle and 0 otherwise.

Figure 9.16 shows the distribution of training set input size in positive (same particle hits) and negative clusters. The size of negative clusters is balanced while the most common positive cluster is 4. Larger positive clusters are less frequent which reflects the actual proportions of the dataset, i.e. particle tracks form a small fraction of the overall dataset.

The cluster consistency model is a 3 layer neural network that uses all the features available from the detector: global coordinates (x,y,z), identifiers of the barrel endcap and disk layer, identifiers of the modules in eta and phi, inner angles of the pixels and the norms of the surfaces. After optimization of the model through a grid search, the number of units per hidden layer

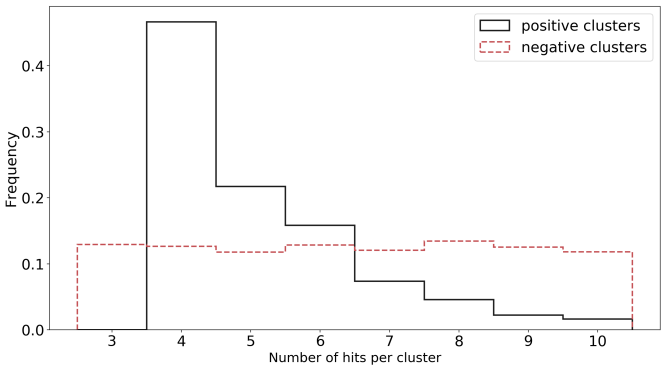
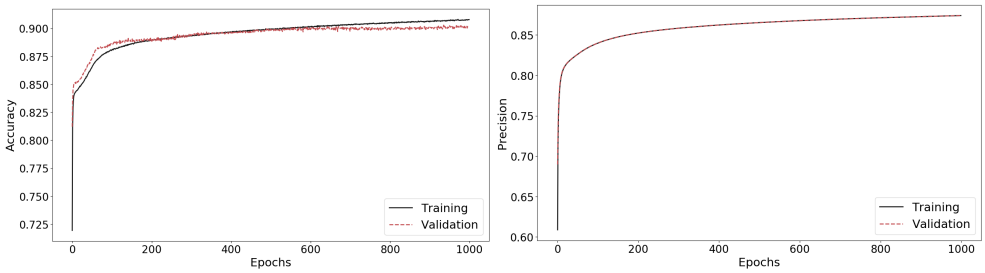


Figure 9.16: Distribution of the input size in positive and negative instances

is fixed at 200 with a tanh activation function. The output is a single unit with a sigmoid activation function. The training dataset is designed to be balanced. It is scaled using a standard scaler (subtracting the mean and scaling to the variance). The validation set is formed by ANN buckets that will contain varying leading particle sizes (unbalanced validation). Contrary to most models presented in this work, the cluster consistency classifier requires a relatively large number of epochs before reaching its peak performance. Figure 9.17 shows the training and validation accuracy in (a) and the precision[8] in (b) as a function of the number of epochs.



(a) (b)

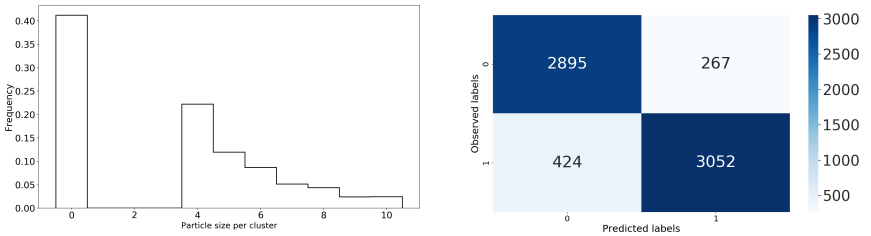
Figure 9.17: (a) Accuracy and (b) precision of the cluster consistency model over the epochs.

In Figure 9.17(a), we can clearly see the model *learning* until approximately epoch 500 where a slight overfitting starts to take place. We use early stopping to select the appropriate number of epochs during training. The accuracy of the model on unseen clusters is of 89% at epoch 500 while the precision reaches 87%. These values are computed on the validation set presented in Figure 9.18(a). When selecting a set of hits from ANN buckets, the most common case is the one where no track is found. With an accuracy of 90%, the majority of negative clusters are correctly identified as negative while a precision at 87% means that most predicted positive clusters are true tracks (seeds). Figure 9.18(b) shows the confusion matrix of the model on the validation set.

9.5 Particle finding with TrackNet and DTL

With a defined distance function and a model accurately deciding whether an arbitrary sized set of hits form a consistent cluster, the next step is the merging

[8]Precision in statistics is equivalent to purity in particle tracking.



(a) (b)

Figure 9.18: (a) Particle size distribution of the validation set. (b) Confusion matrix of the model.

of hits into tracks. This process starts with the computation of the pairwise distance between hits in a mapped bucket. The closest neighbors are merged into clusters of incremental size, i.e. a cluster of 3 hits is formed then the closest hit added to form a 4 hits cluster and this process repeats until reaching 10 hits. At every stage, the consistency classifier produces on a probability on the quality of the current cluster. The cluster with the highest probability is selected as a final track output. According to this description, overlapping or duplicate clusters can be formed. A duplicate removal procedure is run on top of the output tracks. If clusters with similar quality probability are overlapping, the DTL algorithm selects the one that minimizes the spread in the TrackNet feature space. The spread is the maximal width or height reached by the cluster, i.e. how much does a cluster spread along the new feature space. Algorithm 3 details these steps.

Algorithm 3 The DTL clustering strategy.

Input : List of hits in raw feature space
Output: List of tracks

```
Buckets=ANN(hits,distance=Angular)
MappedBuckets=TrackNet(Buckets,eta,OutputDim)
for mb in MappedBuckets do
    D=PenaltyDistance(mb)
    for N in [3,4,5,6,7,8,9,10] do
        Clusters=GetNeighbors(D,N)
        ClustersProbability ← ConsistencyModel(Clusters)
    end for
    ClusterSpread=Spread(Clusters)
    Tracks ← Clusters[min(ClusterSpread) and max(ClustersProbability)]
end for
```

The $ANN()$ function (detailed in Chapter 5 and 7) creates buckets from the global coordinates of the hits with the angular distance. The $TrackNet()$ function maps hits (in buckets) into a new feature space. The $PenaltyDistance$, defined in Eq. 9.7, returns a pairwise distance matrix D where $D[i, j]$ contains the penalized euclidean distance between the hit i and the hit j. The $GetNeighbors(D, N)$ selects N closest neighbors from each hit in the D matrix. The hit and its neighbors form a cluster. The $ConsistencyModel()$, introduced in Section 9.4.2, produces a probability per cluster of size N. $Spread()$ returns the maximal range of the cluster. It is the $max(\sum_{u=0}^{d} max(p_u) - min(p_u))$ where d is the dimension of the TrackNet output and p_u the u^{th} feature in the learned space. The track that maximizes the consistency model probability while minimizing the spread is retained for the final output.

Figure 9.19 shows the final tracks size distribution as returned by Algorithm 3. As expected, the distribution follows the leading size particle distribution in ANN buckets. The tracks found by the DTL are a subset of the ones contained by ANN buckets and more specifically the ones with at least 4 hits (as defined in the DTL algorithm).

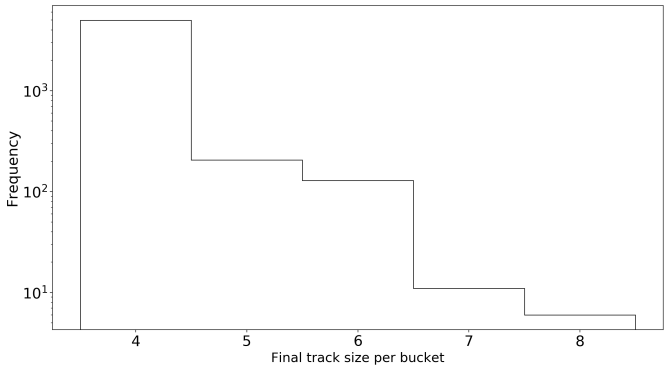
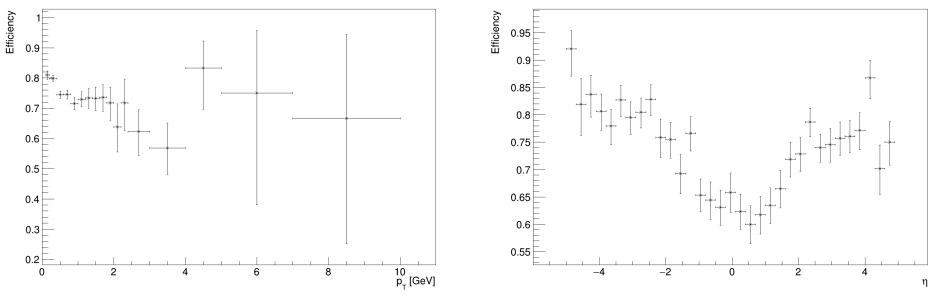


Figure 9.19: Retrieved track size after TrackNet mapping and DTL clustering. Only unique and highly ranked tracks are selected.

With the current minimum input size of the DTL clustering as well as the bucket size of 20 hits, the tracks found are mostly seeds of 4 hits or more. The TrackNet+DTL strategy takes as input ANN buckets that can be viewed as fast binning of the dataset. Increasing the size of the input buckets will allow the capturing of larger tracks that are in turn reconstructed by the DTL clustering.

Figure 9.20 shows the efficiency of the TrackNet-DTL strategy as a function of P_T in (a) and as a function of the pseudorapidity η in (b). The efficiency is computed on the tracks *contained* in the ANN buckets. If a track is not contained (by 4 hits or more), it is not represented in this evaluation.



(a) (b)

Figure 9.20: Efficiency of the TrackNet+DTL approach versus the transverse momentum P_T and the pseudorapidity η.

In order to obtain the efficiency along the pseudorapidity range, the mappings of 4 TrackNet models are combined, i.e. one model per pseudorapidity bin. As discussed in the TrackNet evaluation section, the performances vary depending on the pseudorapidity bin. For example, the mapping performances are the lowest in the central region ($|\eta| < 1$) and this is reflected in the final efficiency after applying the DTL clustering.

We are now interested in evaluating the impact of the penalty term in the clustering distance computation. As previously shown in Eq. 9.7, the penalty is an additional weight in the pairwise euclidean distance when two hits are

physically incompatible in forming a track. Figure 9.21 shows a comparison of the efficiency when keeping or removing the penalty term in the pseudorapidity bin of $2 < |\eta| < 3$. The impact of the penalty weight is significant: on average, the efficiency gain is of 10%. When keeping wrong pairs of hits (no penalty), clusters are formed around them instead of around compatible pairs. This results in more fakes and less true positives since only *the* best cluster is selected by the consistency model. The exclusion rules implemented by the penalty function are therefore not learned by the consistency model and enforcing them into the pairwise distance *helps* the model focus on more relevant combinations.

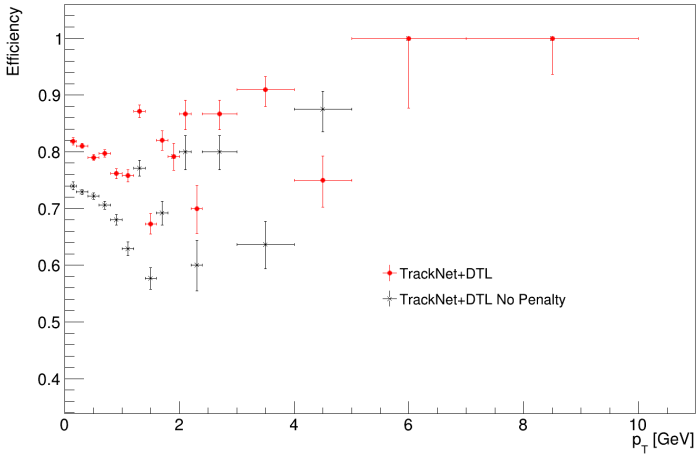


Figure 9.21: Impact of the penalty term on the tracking efficiency. The efficiency is a function of the transverse momentum P_T.

9.6 Summary and future directions

In this chapter we presented a first end to end machine learning approach for particle track reconstruction designed and applied on ATLAS Phase-II Inner Tracker events. This is a particularly complex task that required the design of novel metric learning model as well as a novel clustering algorithm. The unique feature of the TrackNet model is its "tracking aware" loss function. The loss function shapes the output into a clustering ready space. However, due to the nature and strong presence of noise hits in the dataset, the clustering cannot be a trivial euclidean distance grouping. The Dynamic Tracking Linkage (DTL) is a clustering algorithm that uses a penalized distance function as well as a trained model to detect and select the consistency of a cluster.

The results showed a high efficiency with respect to the input content (ANN buckets). The first improvement that can be considered is an increase in the input size with larger ANN buckets or binning. The TrackNet model has to be retrained on larger inputs and the consistency classifier can either be kept the same (with an output of size 4 to 10) or retrained with a larger output range (output of size 4 to 20 for seeds and 10 to 30 for tracks for example).

TrackNet+DTL has been tested in the pixel sub-detector. In order to extend the testing to the full detector, either a retraining including the Strip sub-detector is necessary or the development of (an additional) Strips TrackNet model amounting to one model per sub-detector and per pseudorapidity bin (approximately 8 models in total).

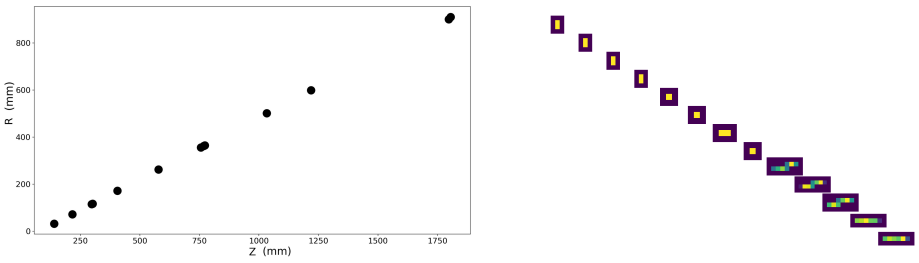
The consistency model in the DTL clustering is crucial. It is able to correct wrong clusters formed on the TrackNet output. The more accurate the model

is, the less false positives and fake tracks are returned. It is possible to improve the consistency model by including additional features: cluster images. The next Chapter describes a convolutional network that learns the quality of ANN buckets from the cluster images.

10

CS-Conv: Convolutions on the Cluster Shape

The interaction of a particle with the detector consists of more than module coordinates. At every interaction, a number of pixels can be *activated* through the charge lost by the particle. As a result, every hit in the detector can be seen as a 3D image where the pixels correspond to the activated pixels and their color to the deposited charge. The number and shape of the these images encode important information on the hit. In Chapter 7, the metric learning model is trained on inner angles computed from the pixels in the image. These angles contributed significantly to the high accuracy of the model. In Chapter 8, due to the ATLAS Phase II Inner Tracker (ITk) layout and the new pixel resolution, these angles did not yield similar impact. The inner angles represent a *description* of the information encoded in the full cluster images. In this Chapter, we will explore the use of Convolutional Neural Networks (introduced in Chapter 4) to extract all the information encoded in the cluster shape of the hits. Although this technique is tested on both the TrackML dataset and the ITk dataset, the key results are shown for the ITk dataset.

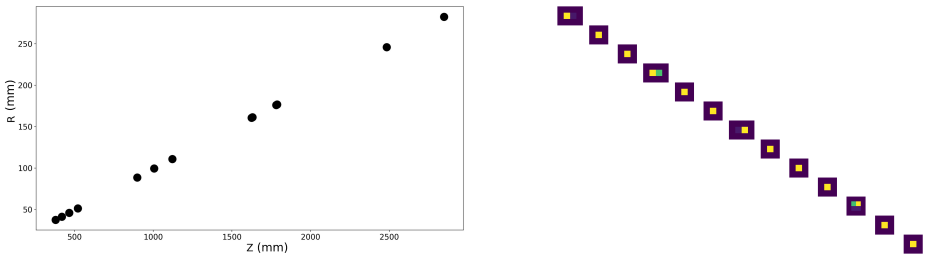


(a) (b)

Figure 10.1: TrackML dataset, an example of a 13 hit particle. (a) Longitudinal view of the particle. (b) Images from clusters and deposited charge associated to the hits in (a). The images are shown in arbitrary axis.

Figure 10.1 shows a particle from the TrackML dataset (a) in the R-Z plane and (b) with the corresponding images of the clusters. Each image represents a hit shown in 10.1(a) with consistent order along the Z axis. The colors in the images are proportional to the deposited charge. The 5 inner-most hits belong to the pixel sub-detector (see Section 6.3.1 for the TrackML detector layout). The associated inner-most images have therefore the most granular resolution. Short and long strips contain less information with only one or two modules activated.

Figure 10.2 shows a particle from the ITk dataset. It contains 13 hits as well but contrary to Figure 10.1, only pixel hits are shown. On the left side of the Figure, the particle is displayed along the R-Z plane while on the right side, the corresponding images from the cluster shape of the hits are shown ordered along the Z coordinate (arbitrary axis).



(a) (b)

Figure 10.2: ITk dataset, an example of a 13 hit particle. (a) Longitudinal view of the particle. (b) Images from clusters and deposited charge associated to the hits in (a). The images are shown in arbitrary axis.

In the case of the ITk dataset, the charge is replaced with the Time over Threshold (ToT) information. As a consequence, the colors in the images are proportional to the ToT. The distribution of the ToT values recorded in a single ITk event is shown in Figure 10.3. Since the clusters are considered as images, not only the shape is important but also the spread of the ToT values within a cluster and their correlations. By design of the ITk detector layout, the shapes of the clusters encode less information as compared to Figure 10.1(b). In the reminder of this Chapter, only the ITk dataset is considered.

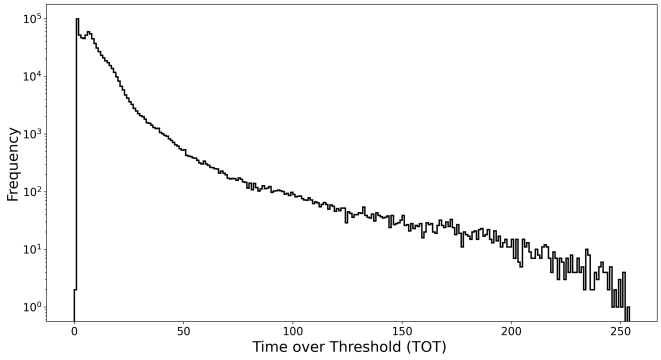


Figure 10.3: Time over Threshold distribution for an ITk event. The ToT is recorded per activated module so one or more ToT values are associated with a single hit.

The idea of this Chapter is to train a CNN on pixel *images* to predict whether a group of hits is produced by the same particle. In the next section, we discuss the techniques and choices made to convert the cluster shape into images and the construction of the training dataset.

10.1 Charge clusters to images

The number of pixels per hit vary greatly. Figure 10.4 shows the distribution of the cluster sizes per hit for an ITk event (approximately 250K pixels). While very large clusters (>100) are attributed to delta electrons (see Chapter 1), the majority of hits are associated with fewer than 20 pixels. In order to generate same size images from the clusters, we propose to translate all clusters into 35x35 centered images. The clusters with less than 35 pixels on either axis will be centered in 35x35 image and completed with zero values. Larger clusters will be cropped (and centered) into 35x35 images as well. A size of 35x35 images is a trade-off to contain the majority of pixels while maintaining a small enough image dimensions. A larger size would result in the majority of images being spare.

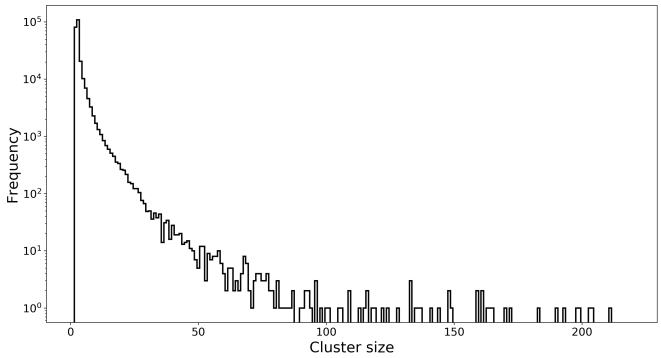


Figure 10.4: Distribution of the number of pixels per hit for one event.

After homogenizing the sizes of cluster images, the second dataset format decision is the number of hits or images to consider as input to the model. The task at being to learn correlation between cluster images, it is important to consider multiple images that likely contain a particle or a seed. This is consistent with Approximate Nearest Neighbors (ANN) discussed in details in this document. The input of the model is therefore a bucket of cluster images. The bucket is defined as a set of neighbor hits using the angular distance and the global coordinates as features. These considerations have been demonstrated to provide high quality bins for the finding of a particle or a seed (four hits or more produced by the same particle). Technically, the input of the model will have the shape (`number_samples`,35,35,`bucket_size`). Images have 35x35 dimensions. The bucket size is a parameter of the approach and can be selected depending on the application intended. In our case, for finding seeds or tracks, buckets of 20 hits are sufficient. For the model however, the different images of a bucket will be represented as image channels similarly to RGB channels of a colored image.

Figure 10.5 shows an example of a 20 cluster images bucket built along the angular distance. No reconstructed track is associated with this bucket. In fact, the fraction of buckets that contain a *reconstructed* track is small compared to the ones that contain only noise hits. This is due to reconstruction cuts applied on the dataset that allow only a fraction of particles to be considered for reconstruction. More precisely, in a typical ITk dataset and considering only pixel hits, only 20% of the hits contribute to reconstructed particles. The remaining 80% are noise hits and therefore are not associated with any reconstruction particle. This results in a heavily unbalanced dataset.

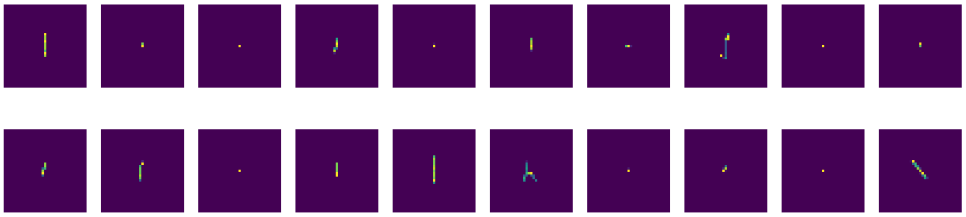


Figure 10.5: A bucket of hit clusters as centered and resized images.

10.2 Dataset balance

As specified earlier, the target of the model will be whether a bucket contains a reconstructable particle or not. This is achieved by labelling the buckets with 0 if no track larger than a certain threshold if found and 1 otherwise. This threshold will be referred to as **`min_hits`**. A standard value for `min_hits` would be 4 but we are also interested in exploring tighter requirements such as values above 7. Higher values allow to find longer tracks. We will analyze the seeding results of the model with a `min_hits`=4 and the tracking results with a `min_hits`=7.

When choosing `min_hits`=7, the fraction of positive values in a random sample from an ITk event is approximately 10%. It increases to 25% with `min_hits`=4. This creates a massively unbalanced dataset. Since the considered dataset is formed by images, we use the undersampling approach.

Undersampling implies the removal of instances associated with the majority class. This is the removal of a subset of noise buckets from the dataset such as the number of positive instances (buckets with tracks) is approximately equal to the number negative instances (buckets with no tracks). Balancing the dataset is only applied to the training set and not to the test or validation set. The idea is that the model *learns* from a balanced dataset but is evaluated on a real (thus unbalanced) dataset.

10.3 CS-Conv model

The CNN used in this application has the standard architecture of sequential convolution layers, pooling and normalization layers and, at the end, a fully connected layer that maps the last weights into a single probability, i.e. the output of the model. The architecture is chosen as follows:

1. The first model contains only one convolution and one dense layers.

2. After each evaluation of the network on a validation sample, an additional hidden (convolution, fully connected or normalization) layer is added.

3. The previous step is repeated until the performances decrease when adding more hidden layers.

After incremental improvement of the architecture of the model, CS-Conv is formed by two convolution layers with a normalizing layer in between. Convolutions are performed with a kernel size of 5x5. A pooling layer is then added followed by a drop-out layer. The output is then flatten and finally a dense layer transforms the output into a a single probability using a Softmax function. Convolutional layers use a Relu function. The architecture is presented in Figure 10.6. Since the task is to classify bucket cluster images,

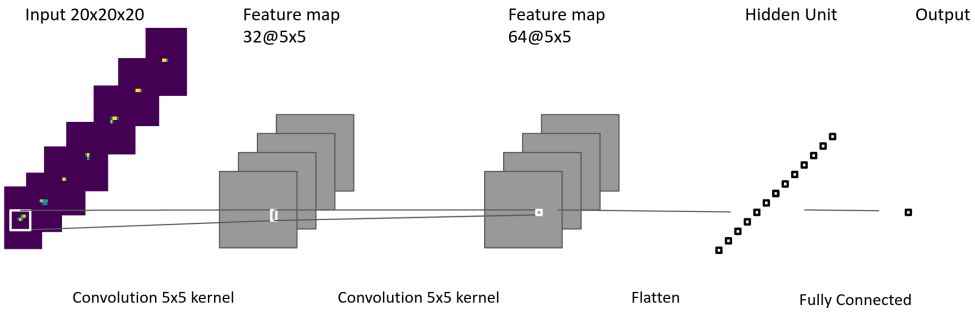
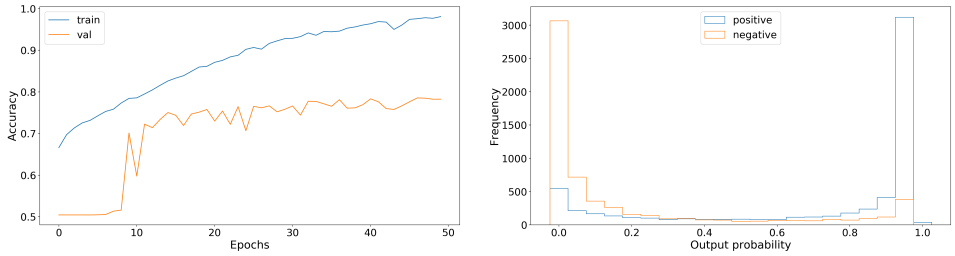


Figure 10.6: CS-Conv model architecture.

we use a categorical cross entropy loss function with an Adam optimizer. The model is trained on all the possible buckets in an event. Each bucket is labelled as positive if it contains at least 7 hits and negative otherwise (`min_hits`=7). The accuracy of the classifier directly translates its ability to decide whether a bucket of cluster images contains a reconstructable particle or not. The evolution of the accuracy on a training set as well as on a validation set is shown in Figure 10.7(a). The model output for positive and negative buckets (on the validation set) is shown in Figure 10.7(b). We can see a clear separation in the model response between a reconstructable bucket and a noise bucket. Positive samples with low probability (<0.5) constitute fake negatives while negative samples with high probability (>0.5) form the fake positives.



(a) (b)

Figure 10.7: (a) Evolution of the training and validation accuracy over the epochs. (b) Distribution of the model output probability for positive buckets and negative buckets on a validation sample (unseen examples).

The model reaches an accuracy of 78% on unseen samples and with a minimum threshold of 7 hits for positive buckets. The training set consists of 37K buckets with a balanced positive/negative ratio. The validation is performed on 10K additional buckets. It is expected that the model improves further when increasing the size of the dataset (to millions or billions buckets). It is interesting to see however that even with relatively low statistics, the CNN is able to extract a pattern from the cluster images only and make a decision on their content with at least 78% accuracy. The cluster images do not contain any information on the detector layout, i.e. which detector region is associated with the cluster. This information is crucial for the reconstruction of tracks given that the detector layout changes as a function of the pseudorapidity. This means that the cluster images will be dependent on the detector geometry and therefore on the pseudorapidity. This substantial improvement of the model is presented in Section 10.4. In the remainder of this section, we discuss two additional aspects of the model : The performance of the model when

lowering the threshold to `min_hits`=4 and the correlation between the model error (fake positives and fake negatives) and the size of the reconstructable particle in a bucket.

The accuracy of the model drops by approximately 10% when the labelling of positive buckets changes from `min_hits`=7 to `min_hits`=4. This is when considering exactly the same architecture and hyper-parameters. It appears that an additional optimization of the model is necessary when changing the label definition.

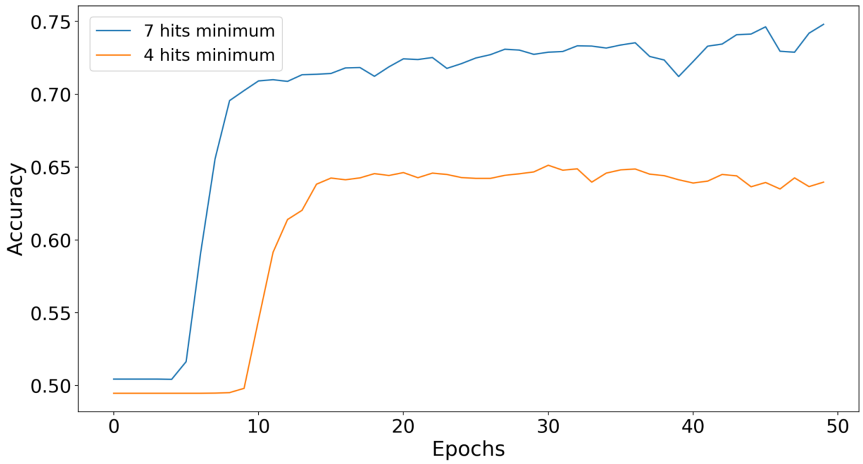


Figure 10.8: Model validation accuracy for a `min_hits`=4 and `min_hits`=7.

Figure 10.8 illustrates the drop in validation accuracy when lowering the minimum track size per bucket. This decrease in performance is however expected since the dataset with a threshold of 7 hits is a subset of the one with a threshold of 4 hits. The learning task becomes therefore more challenging. Figure 10.9 shows the impact of the leading particle size per bucket on the model decision. This result is obtaining when running the model on an unseen event and recording the leading particle size for each bucket. The correlation of the leading particle size and the model response is visible on the figure. The model makes more accurate decisions on buckets containing larger tracks. As a reminder, noise buckets (<4 hits) do not contain any *ATLAS* reconstructable track but can (and in fact with considerable proportions) contain a truth simulated track.

The proportions of buckets with no reconstructable track (size 0 in the x-axis of the figure) but with a simulated truth track are also highlighted in Figure 10.9. The black squares that overlay some of the pixels in the figure, denote that more than 50% of the buckets represented by the pixel contain a truth track with at least 4 hits. It is interesting to notice that most of these boxes are associated with a high model output. This means that although the model presents false positives, i.e. no reconstructable track and a positive probability >0.5, at least half of these contain a truth track.

10.4 Upgraded CS-Conv

Upgrading the CS-Conv model is done following two axis : 1/ A model per pseudorapidity region and 2/ Geometry features as well as global coordinates of the hits are considered via a binary and a multi-class classifier. Figure 10.10 summarizes these upgrades. The global model, now referred to as VCS-Conv (Voting Cluster Shape Convolutions), takes as input buckets of hits queried from a single pseudorapidity region. The cluster information is translated

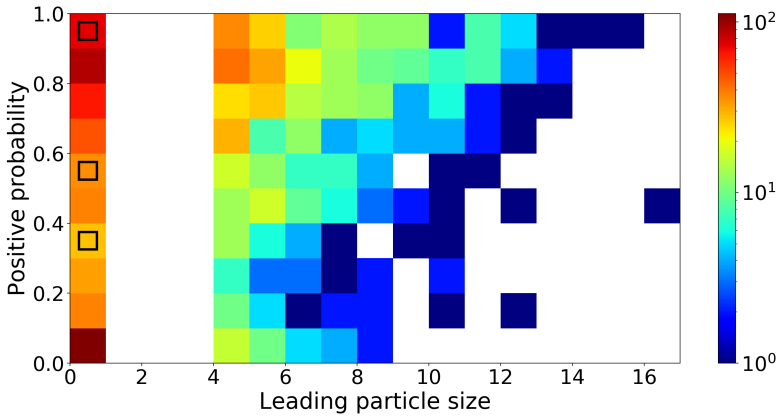


Figure 10.9: Impact of the leading particle size on the model decision. Positive buckets contain at least 7 hits produced by the same particle. Black squares denote the presence by at least 50% of truth tracks in noise buckets.

into 20x20 images and passed to the CS-Conv. The raw information consisting of global hit coordinates (x,y,z), module coordinates (barrel endcap and layer disk) as well as cluster summary features are passed to a binary and a multiclass classifier (separately). The summary features include the following values: the number of modules in the cluster, the maximum ToT value within the cluster and the average value (Refer to Figures 10.3 and 10.4 for the distributions of these values).

Both the CS-Conv and the binary classifier output a probability of containing a seed (or track depending on the threshold considered). The multiclass classifier is trained to return the category of the leading particle:

- 0 if no reconstructable particle is contained.
- 1 if the leading particle size is between 4 and 6.
- 2 if the leading particle size is between 6 and 9.
- 3 if the leading particle size is larger.

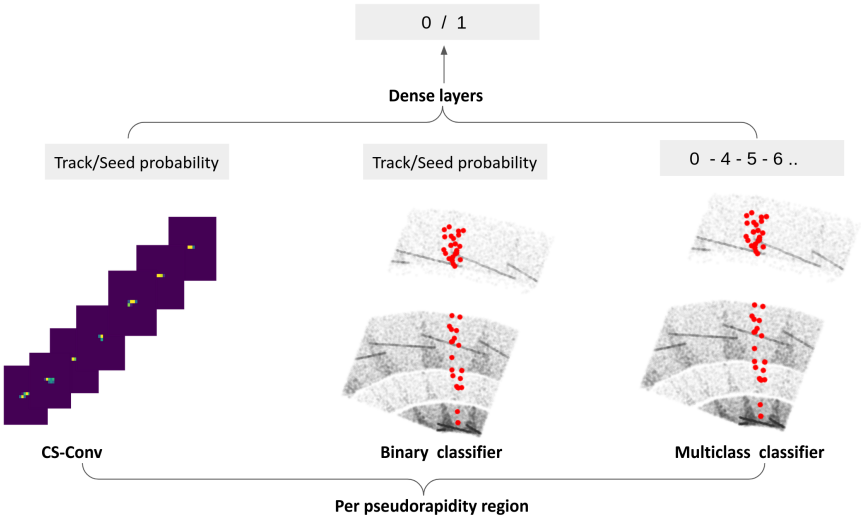


Figure 10.10: Combination of multiple models : CS-Conv on cluster images, binary and multiclass classifiers on raw hit information.

It is possible to train the model on the actual leading particle size rather than its category. The model would choose from a set of 16 values (values $<$ 4 are always classified as noise). Such a model would require a much more complex structure compared to a model with fewer output values. Moreover, discriminating between a bucket that contains a 7 hit track and one with 8 hits has only a minor impact on the global goal of the proposed approach. As a result, the multiclass classifier will have 4 distinct output when `min_hits`=4 and 3 distinct output when `min_hits`=7. The decisions of the trained three models are later combined in a dense layer to *vote* for the correct output.

The first step in the voting process is to optimize the *raw* features classifiers individually. Two pre-processing choices are made before any training: Balancing of the dataset through over-sampling and the scaling of the different features. Over-sampling involves the *production* of synthetic examples from the minority class (buckets with tracks in our case) such as the number of positive and negative examples is approximately the same. A wealth of techniques are available to perform over-sampling on an unbalanced dataset [1]. For the classification of buckets, we propose to use Adaptive Synthetic Sampling (ADASYN) [2] since the core idea behind it aligns well with the nature of hit buckets achieving therefore superior results to any other over-sampling algorithm. In ADASYN, every sample from the minority class produces a number of synthetic examples. As the authors describe it, the number of synthetic examples per minority sample is dependant on the complexity of the sample (hence the *adaptive* characteristic). To determine the complexity of a sample, ADASYN evaluates the proportion of majority class examples in the neighbourhood of the sample. The idea is the following: a sample from the minority class *surrounded* by samples from the majority class is much harder to classify. The classifier needs to see many more examples of it. Once the number of synthetic examples is determined, new instances are constructed as follows[2]:

$$s_i = x_i + (x_{zi} - x_i) \times \lambda$$

Where s_i is the synthetic example, x_i the minority example that is oversampled, x_{zi} a randomly selected minority example from the neighborhood of x_i and λ a predefined weight between 0 and 1. The difference $(x_{zi} - x_i)$ represents the euclidean distance between the two samples in the d dimensional space. d is the number of dimensions of each sample from the dataset (or number of features).

Figure 10.11 shows an example of a generated bucket projected to the R-Z plane. The bucket is shown along an original bucket that contains a 7 hit track. The euclidean distance between the synthetic bucket and the closest original bucket is of 0.5 (along all features). The bucket shown in the figure however has a 1.7 distance to the original bucket. This choice was make to highlight the impact of the ADASYN algorithm along the R-Z coordinates.

After applying the ADASYN oversampling, the number of positive buckets and negative buckets is the same in the case of the binary classifier. For the multiclass model, the oversampling also generates an equal number of example in each category. The next step is to scale the dataset and ensure that all examples have the same variation range. After several trials, we choose the Robust Scaler technique [3]. Instead of removing the mean and scaling to unit variance (as it is done in standard scaling), the Robust Scaler removes the median and scales according to the quantile range. This simple difference makes the scaler *robust* to outliers. More specifically, we use the interquantile range between the 25th quantile and 75th quantile. The classifier is then trained on a balanced, scaled dataset.

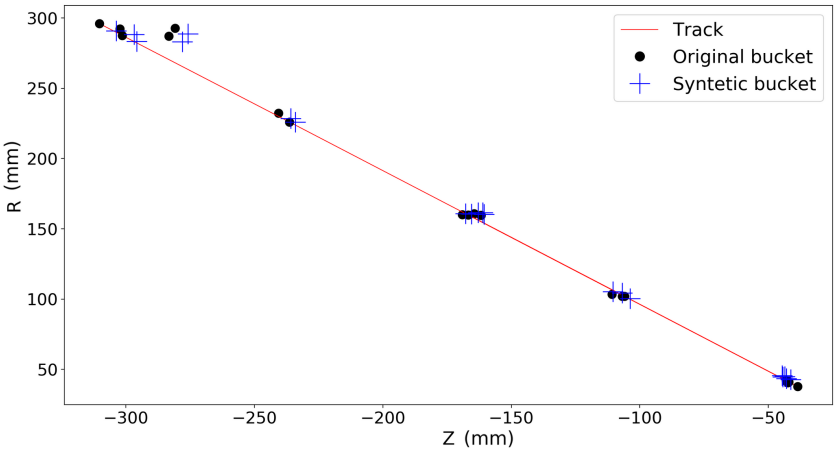
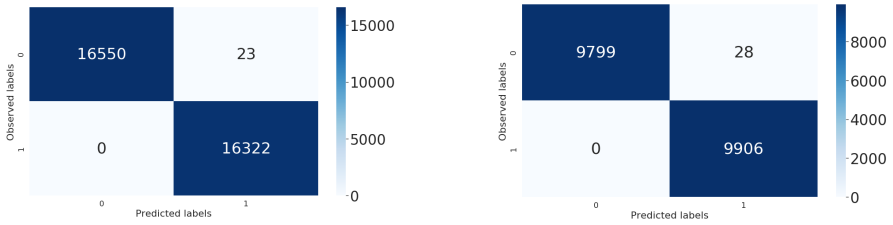


Figure 10.11: Example of an ADASYN synthetic bucket in the R-Z coordinates.

For the classification tasks, we propose to use a voting classifier. The final predicted label, both for the binary and multiclass case, will be the majority one between : A logistic regression (LR), a random forest (RF) and a Support Vector Machine (SVM).

The performances of the voting classifier are summarized by a confusion matrix obtained from the predictions of unseen validation sample. Each confusion matrix shows the results of a single pseudorapidity region and decision mode : binary or multiclass. With the confusion matrix, we can extract the fake positive and fake negative rates as well. A perfect classifier fills only the diagonal ([0,0] and [1,1] pixels). Figure 10.12 shows the confusion matrix for buckets sampled from $|\eta| < 1$ bin with a binary classifier in (a) and a multiclass one in (b). The overall accuracy of the different models is much higher than the CS-Conv model since it relies on all the information collected from the detector.

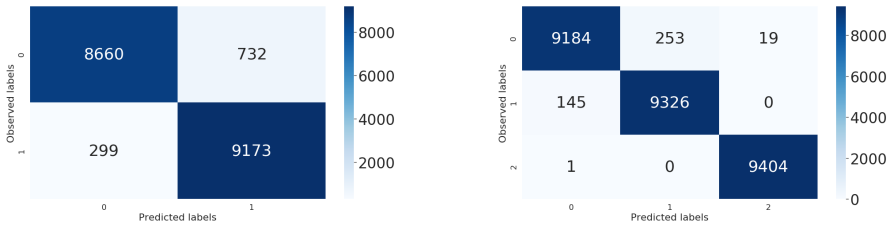


(a) (b)

Figure 10.12: Confusion matrix of the (a) binary decision model and (b) the multiclass model. The pseudorapidity region considered is $|\eta| < 1$ with a `min_hits`=7.

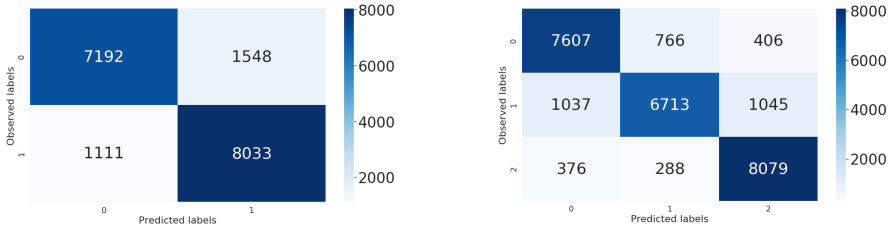
For each pseudorapidity region, the decision produced by the binary classifier, the multiclass classifier as well as the CS-Conv model is combined into a single value through a fully connected network structure. The input is a three dimensional vector, one hidden layer is used to combine the decisions and a single node is used as global output.

The global model (VCS-Conv) combines the decision of the three models through a single hidden layer consisting of a 100 node with a Relu activation function. The output is obtained with a Sigmoid function. We use the mean squared error loss with the Adam optimizer at a learning rate of 10^{-4}. As



(a) (b)

Figure 10.13: Confusion matrix of the (a) binary decision model and (b) the multiclass model. The pseudorapidity region considered is $1 < |\eta| < 2$ with a `min_hits`=7.



(a) (b)

Figure 10.14: Confusion matrix of the (a) binary decision model and (b) the multiclass model. The pseudorapidity region considered is $2 < |\eta| < 3$ with a `min_hits`=7.

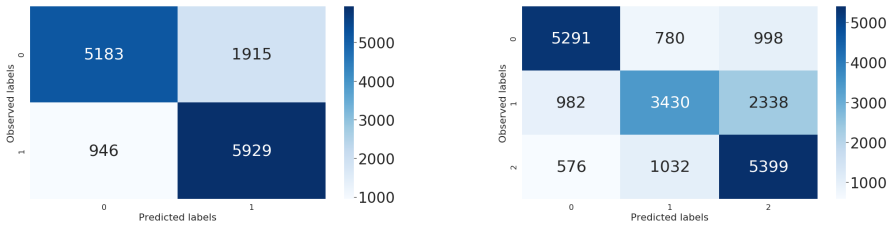
previously established, the dataset consists of the output probability of the CS-Conv, the binary classifier and the multi-class classifier, producing therefore a three dimensional dataset. Under-sampling is used to balance the positive and negative ratios. The result presented next focus on the central pseudorapidity region with $|\eta| < 1$.

Figure 10.16 shows the model output probability distribution for the three initial models : CS-Conv, Binary classifier and Multiclass classifier and the global model VCS-Conv that combines their decision to produce a more robust prediction. It is apparent that while the three input models have a wider range, the VCS-Conv model (black discontinuous line) tends to produce narrower probabilities with the lowest prediction around 0.15 and the highest at 0.90. VCS-Conv is trained to make predictions that are as accurate as the best input model or better if the error of the best classifier can be compensated by the rest of the models.

The ROC curves associated to the models are shown in Figure 10.17. It is the evolution of the true positive rate as a function of the false positive rate. A random classifier would produce a diagonal with as many false positives as true positives (highlighted as discontinuous back curve in the figure for comparison). The VCS-Conv model leverages the different models strength and weakness to produce a more robust decision considering cluster images and raw hit features. VCS-Conv outperforms the three input models along the full range of false positives.

Starting from a false positive rate of approximately 30% (0.3 in the x-axis), VCS-CNN and the binary classifier predictions are similar. We can also notice that at approximately the same false positive rate (22%), CS-Conv achieves a better performance than the multiclass classifier and ultimately reaches the true positive rate of the VCS-Conv.

The goal of this chapter was to propose a method that extracts relevant information from the collected data including images of activated clusters, ge-



(a) (b)

Figure 10.15: Confusion matrix of the (a) binary decision model and (b) the multiclass model. The pseudorapidity region considered is $3 < |\eta|$ with a `min_hits`=7.

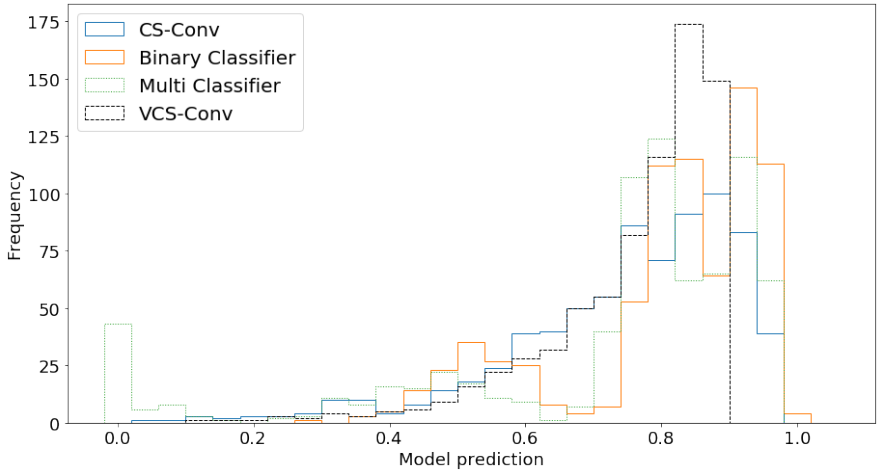


Figure 10.16: Distribution of the different models output probability as well as the global VCS-Conv model.

ometry information, global hit coordinates, inner cluster angles as well as the coordinates of these clusters. We proposed to train a convolutional network as well as binary and multiclass classifiers on this heterogeneous dataset. Moreover, the output of the three models is combined in a dense neural network to predict the most likely label : presence or absence of a track in a set of hits. This approach resulted in a high accuracy and precision. An extension of this work would be to consider a much larger dataset (thousands or hundred thousands events) in order to further validate the performances and potentially extract interesting correlations between the different models.

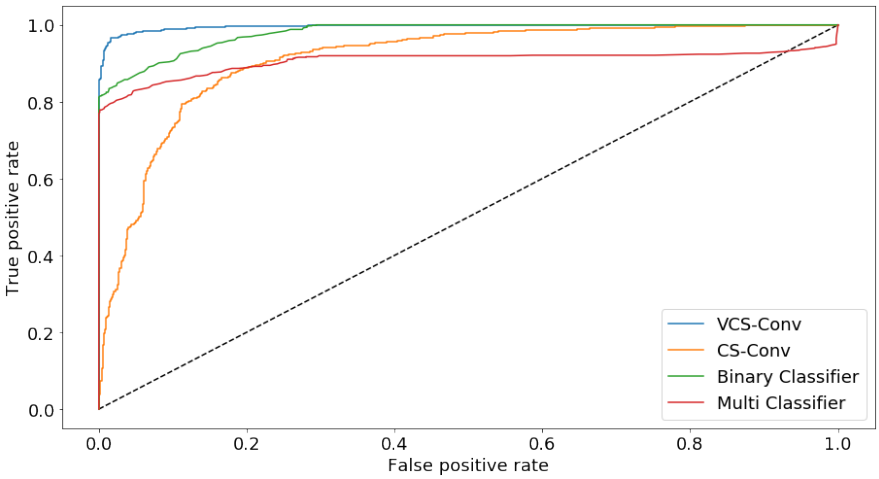


Figure 10.17: ROC curve of the different models as well as the global VCS-Conv model.

Bibliography

[1] Chawla, Nitesh V. "Data mining for imbalanced datasets: An overview." Data mining and knowledge discovery handbook. Springer, Boston, MA, 2009. 875-886.

[2] He, Haibo, et al. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning." 2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence). IEEE, 2008.

[3] Robust Scaler on Scikit-learn `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`

[4] Rokach, Lior. "Ensemble-based classifiers." Artificial intelligence review 33.1-2 (2010): 1-39.

Conclusions and Future Outlook

A slow but fundamental shift is taking place across high energy physics experiments. From theoretical modeling and performance optimization to the search for new physics, machine learning models are at the core of many large scale on-going studies. While not yet relied on in the ATLAS framework, a large number of ML projects that were implemented and tested around "toy" datasets are either currently or in the near future being confronted to actual ATLAS simulation datasets. An expected consequence is that many of these novel models will outperform current algorithms either in speed, accuracy or both opening new solid research directions.

This thesis details such a journey, starting from an idea to rapidly capture particle tracks, growing into a multistage framework optimized on the TrackML dataset and later fully "upgraded" into an end to end ML based tracking software for the ITk detector. Figure 10.18 summarizes the various components of the conducted research and places them along the tracking pipeline (x-axis) and along the readiness metric (y-axis).

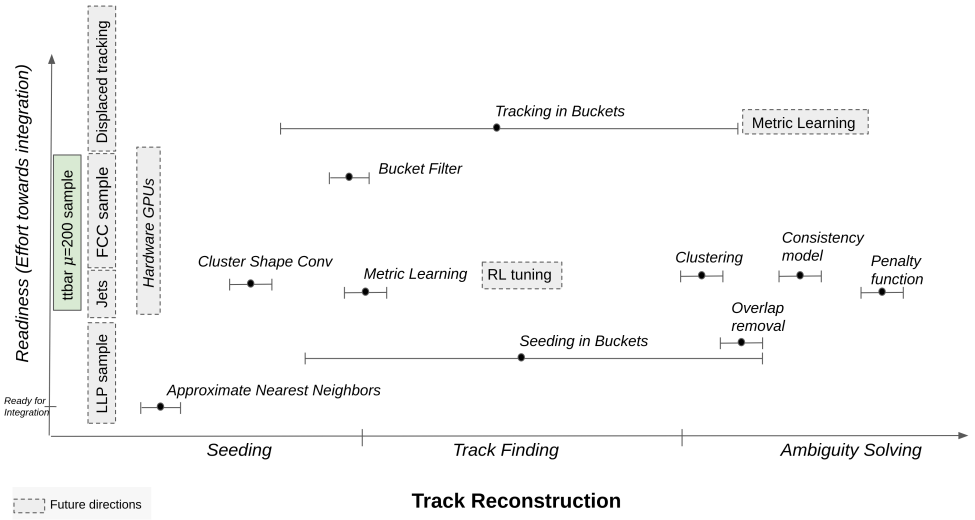


Figure 10.18: Overview of the thesis. Implemented models and future research directions as a function of their positioning along the tracking chain and their readiness for use.

The readiness metric allows to place the different techniques and models as a function of their performances and their ease of integration into tracking frameworks. As such, a technique that is at least as efficient as the standard tracking requirement and that is easily portable or already available for use in a tracking framework (ATLAS or ACTS) is placed closer to the x axis. The various models are also placed along an *approximate* tracking pipeline starting

with seeding to track finding and ambiguity solving. Although the proposed models do not necessary fit the standard tracking chain, a parallel can easily be made where algorithms that bin the data fall closer to seeding while models that learn a tracking feature space resemble track finding and finally algorithm that define the boundaries of tracks and evaluate overlaps are placed closer the ambiguity solving.

Interesting future extensions are denoted by grey discontinuous boxes and error bars along the x axis represent the range of the techniques in terms of standard tracking. As a result, event processing techniques are placed closer to the seeding stage while algorithm that process tracks are located closer to the ambiguity solving. Approximate Nearest Neighbors (ANNs) allow to split an event into buckets of hits that likely contain tracks. Various techniques have been evaluated in this thesis and selected libraries provide C++ APIs making them ready to use in any tracking framework. Combining standard tracking and seeding with ANNs resulted in high efficiencies (as compared to standard requirements) and demonstrated the feasibility of restricting the seeding to a reduced number of hits. To obtain these results, the various components where already tested in the ATLAS standard tracking framework but do require additional integration to form a standalone tracking software. The metric learning model and the clustering algorithm with the consistency model and the penalty function were implemented and evaluated in a standalone python framework. Although the necessary effort to integrate a trained network into any software is negligible, the overall efficiency of the model is still below the standard requirements and is therefore placed further away in the readiness axis. The cluster shape convolutional network demonstrated superior accuracy as compared to the alternative model relying on raw data only. Similarly to the metric learning model, although its integration is straightforward and the accuracy of the model is higher than required, an extensive validation on a very large number of events is necessary in order to be used as an alternative input in standard tracking. The bucket filter as well as the standard tracking restricted to ANN buckets resulted in low efficiency as compared to standard tracking. An interesting extension is to evaluate tracking in buckets after mapping with a metric learning model.

In the following, the various building blocks of the approach, from data to evaluation, are summarized.

Data and hardware directions

The data used in this research focused on ttbar collisions events with a pileup of 200. An interesting extension is to investigate different physics signatures and pileup such as a Long Lived Particle (LLP) data sample, a displaced tracks sample, a dense environment (Jets) sample, a $\mu = 1000$ sample such as the one expected at the Future Circular Collider (FCC). Each data sample would challenge different properties of the proposed tracking approaches and could, with some iterations, show potential.

While ANNs were demonstrated for GPUs, all of the models presented in this work have been evaluated on CPU only. It has been established by now that most machine learning models can be significantly accelerated on hardware (GPUs and FPGAs). A very promising direction is the deployment of the proposed framework on hardware especially that every component presented in this thesis was intentionally designed and optimized for a multi-threaded execution. N ANN buckets can be generated, mapped with a metric learning model and clustered in a completely independent workflow. With this acceler-

ation, the approach can also be considered for triggering where reconstructed tracks of electrons or muons are required (to complete the calorimeter information) for their identification.

From events to buckets

The first contribution of this work is the use of Approximate Nearest Neighbors for fast particle tracking. Building trajectories with ANNs is a novel idea within and outside high energy physics. It is intended as the first step to digest and navigate physics events for tracking purposes. ANNs build data-driver sets of neighbors where a particle track is most likely found. Contrary to binning, which is the closest alternative in standard tracking, ANNs return a set of neighbors *depending* on the event data structure and use any defined similarity measure (including learned functions). This completely alleviates the problem of having arbitrary large bins in dense detector regions and allows to *bend* the bucket according to the considered similarity function. While in this thesis ANNs have been combined with metric learning to project the dataset into a new feature space, it is interesting to envision an extension where the search function would iteratively work with the metric learning function to jointly map and search hits.

For applications where a bucket of hits is not mapped through a similarity model, it is necessary to use a filtering. Filters are binary classifiers that learn to distinguish between a bucket that contains a track from one that does not. In this thesis, we propose filters based on raw information of the hits as well as on heterogeneous input : raw features and cluster images (CS-Conv). More complex filters can be investigated where the task is not only to distinguish buckets but also to classify the hits within the buckets into particles or noise. Another interesting topic is the evaluation of the query position impact. Both extensions have been proposed and studied in master projects in our research group.

Metric Learning

The second major contribution of this thesis is the use of metric learning for particle tracking. Metric learning uses the supervised information obtained through simulation to map raw hits into a new feature space where particles are easily retrieved. Standard metric learning techniques (LFDA) were used on the TrackML dataset while a novel design was proposed for the ITk layout. While ANNs combined to LFDA achieved high performances on the TrackML dataset, the approach could be further improved by training a model per pseudorapidity region. The shift from TrackML detector to the ITk represented a significant transition in this work due the dataset differences discussed in this document. The complexity of the dataset allowed to conceive a novel metric learning model termed TrackNet. While the overall performance of the model is currently below the standard requirements in tracking, its design concepts and implementation allow a variety of improvements. The loss function is proposed to translate tracking constraints into the deep learning model with an adaptable structure easily augmented with additional terms. Moreover, a single TrackNet model is proposed for each pseudorapidity region with the selected optimal parameters. Interesting extensions include Neural Architecture Search (NAS) for the optimization of the architecture, Reinforcement Learning (RL) for the automatic learning of all the parameters (including ANN parameters such as the bucket size) and the inclusion of cluster shape features.

Fast Search in Standard Tracking

Another contribution of this work is the evaluation of the ATLAS standard tracking algorithm in ANN buckets. The first prototype ran the full tracking algorithm in buckets of 50 hits. This approach was successful in retrieving particles only when they are fully contained in the buckets. While this first trial demonstrated the feasibility of running standard tracking in small bins, it also performed only as good as the bucket quality (which lowered from TrackML to ITk). The second prototype using standard tracking used ANN buckets only for the seeding stage with displaced buckets to capture tracks with an important Z0 offset and an overlap removal procedure. This approach allowed to find all the tracks of interest in an event without penalizing the speed of the tracking. It requires only 5ms to build a track in a bucket with a number of additional low P_T tracks resulting from the loosening of the seeding cuts. An interesting improvement is to use metric learning prior to the buckets sampling. This could not only improve the quality of the buckets but also the performance of the filtering reducing the total number of buckets. The ANN bucketing for seeding can be directly integrated into a tracking software. Although a full multi-threaded integration into a tracking framework is necessary to assess the speed-up factor, the various tests (especially on GPUs) show a considerable speed-up potential. In this context, the ACTS framework appears as the most adapted tracking software for the integration of both the similarity search and the TrackNet model.

Clustering

The Dynamic Tracking Linkage (DTL) clustering, unlike standard clustering algorithms that use static merging constraints, automatically chooses the ideal cluster using prior learned knowledge rather than geometrical rules. The DTL algorithm further uses a penalty function to exclude pairs of hits that violate tracking constraints. This function can be easily enriched with additional terms likely resulting in an improved clustering. An important improvement could be to augment the consistency model with physics driven knowledge. While the current proposal is already "adaptive" by learning from data, it is still *unaware* of detector layout or the physics meaning of the hits. The model can therefore be mutated into a learned Kalman filter.

Machine learning models thrive in massive datasets and each model presented in this research can immensely benefit from training on billion hit, labelled dataset.

Acknowledgements

This thesis has been a true adventure with many sacrifices (I partly worked in computer vision), dramatic twists (obtained a grant and quit my job) and hectic changes (including 6 months of weekly flights between Geneva and Vienna). At the end, each of these had a significant positive impact on both my work and my life. Many people made this journey possible, on so many different levels. I hope I will not leave anyone out.

This work would not have been possible without my thesis advisor Prof. Tobias Golling. Thank you for giving me the opportunity to join the ATLAS group at Geneva university. Thank you for all the fruitful discussions, advice and for your flexibility. Your precise questions and comments at every meeting helped me become the researcher I am. I am also delighted that our collaboration went beyond particle tracking and into the organization of a successful hackathon and a first data science collaboration with the Vienna University.

I would like to thank Andreas Salzburger for answering my email on Feb 2015, nearly two years before the official start of this thesis. Thank you for being enthusiastic about machine learning, already a while ago. Thank you for all the tracking notions you passed on to me. And finally, thank you for always seeing the big picture and providing guidance accordingly. This work would not have been the same if I sat anywhere else than outside your office.

I am thankful to Moritz Kiehn for his feedback and constructive input all along this work.

Many studies in this work would not have been possible without the talent and expertise of Noemi Calace. Thank you for providing and directly helping with the Standard ATLAS Tracking. I enjoyed working with you and learned a lot. Thank you also for being such a good friend and for providing support in difficult times.

I would like to thank Prof. Claudia Plant for all her feedback on clustering techniques. It was a renewed pleasure to learn from you and build on your input.

I would like to thank all the present and past members of the ATLAS-UniGe group for all the fruitful discussions.

I would like to thank my former supervisor Benedikt Gollan and CEO Peter Bruck at Research Studios Austria for allowing me to work part time and maintain a good balance on my thesis. Thank you for the mentoring and encouragement. What started as a regular job turned out to be one the best growth opportunities in my life.

A special thanks to Pauline Gagnon whom I first met (I was an undergraduate) in 2013 during the CERN open days where she told me that yes it should be possible for you to do your master thesis at CERN. When reaching out to her shortly after, she started asking researchers and group leaders about such opportunities (after knowing me for 2 minutes). Thank you for believing in me and in the equality of such opportunities.

I would not have been able to start this thesis without the support of my best friend, my lighthouse Dalila Salamani. We crossed the Mediterranean

sea determined to pursue a PhD at the intersection of physics and machine learning. Thank you for encouraging me during my moments of doubt, thank you for being my office mate and my homeland away from my homeland. Thank you for sharing every step of this journey with me.

Thank you to my family in this continent: my brother Faouzi. Thank you for your support and advice.

I cannot thank enough my parents for their support and their strength. Thank you for believing in me and for trusting me to take risks. Thank you for your courage, your patience... in the distance.