

JOHANNES GUTENBERG-UNIVERSITÄT MAINZ

MASTERARBEIT

Jet Reconstruction in the ATLAS Level-1 Calorimeter Trigger with Deep Artificial Neural Networks

Autor:

Bastian Schlag

Gutachter:

Prof. Dr. Volker Büscher

Prof. Dr. Michael Wurm



JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Masterarbeit in Physik

vorgelegt dem Fachbereich Physik, Mathematik und Informatik (FB 08)

der Johannes Gutenberg-Universität Mainz

am 07. August 2018



Declaration

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Mainz, den 07. August 2018

Bastian Schlag

Contents

1	Motivation	1
2	The Standard Model of Particle Physics	3
2.1	Fundamental Interactions and Particle Content	3
2.2	Quantum Chromodynamics at Hadron Colliders	5
2.2.1	The QCD Lagrangian	5
2.2.2	Asymptotic Freedom and Confinement	5
2.2.3	Hadronic Jet Production	6
2.3	Beyond the Standard Model	8
3	The ATLAS Experiment	9
3.1	Experiments at the Large Hadron Collider	9
3.2	ATLAS Coordinate System and Definitions	10
3.3	Detector Components	11
3.3.1	Inner Detector and Central Solenoid	11
3.3.2	Calorimeters	12
3.3.3	Muon Spectrometer and Toroid Magnet System	15
3.4	Trigger System	15
3.4.1	Design and Functionality	16
3.4.2	Phase-1 Upgrade of First Level Calorimeter Trigger	18
3.5	Jet Reconstruction at ATLAS	20
3.5.1	The Anti- k_t Jet Clustering Algorithm	20
4	Deep Learning Methods	21
4.1	Deep Feed Forward Neural Networks	21
4.1.1	The Perceptron	21
4.1.2	Multi-layer Perceptron	22
4.1.3	Activation Functions	23
4.1.4	Gradient Descent and Backpropagation	25
4.2	Convolutional Neural Networks	28
4.2.1	Basic Concepts	28
4.2.2	Convolutional Neural Network Architectures	30
4.3	Regularization	32
4.3.1	Underfitting and Overfitting	33
4.3.2	Early Stopping	33

4.3.3	Dropout	34
4.3.4	Parameter Norm Penalties	34
4.4	Training Optimization	35
4.4.1	Batch and Minibatch Algorithms	35
4.4.2	Parameter Initialization	36
4.4.3	Input Data Preparation	36
4.4.4	Hyperparameter Optimization	36
5	Monte Carlo Simulation Samples	39
5.1	Fast Simulation Data	39
5.1.1	Event Generation with PYTHIA	40
5.1.2	Fast Detector Response Simulation	41
5.1.3	Anti- k_t Online and Anti- k_t Truth	45
5.2	Full ATLAS Monte Carlo Data	47
5.2.1	$hh \rightarrow 4b$	47
5.2.2	$gg \rightarrow t\bar{t}$	48
5.3	Overview over Available Data and Data Selection	49
6	Jet Reconstruction with Deep Neural Networks	51
6.1	First Approach: Sorted List of Jets	51
6.1.1	Basic Idea and Output Format	51
6.1.2	Network Architecture and Hyperparameters	52
6.1.3	Simple Case Study: Single Jet Reconstruction	53
6.1.4	Multiple Jet Reconstruction	58
6.1.5	Summary of Presented Approach	63
6.2	Second Approach: Regression on Grid	64
6.2.1	Basic Idea and Output Format	64
6.2.2	Network Architecture and Hyperparameters	65
6.2.3	Performance Analysis	66
6.2.4	Summary of Presented Approach	81
7	Adaption to Hardware Resources	83
7.1	General Parameter Reduction	84
7.2	Phase-1 Hardware with Xilinx UltraScale+ FPGAs (XCVU9P)	85
7.3	Phase-2 Hardware with Xilinx UltraScale+ FPGAs (XCVU13P)	87
7.4	Phase-2 Hardware with ACAP	89
8	Summary and Outlook	93
A	Appendix	95
	Bibliography	101

Chapter 1

Motivation

In the upcoming years, the planned upgrades of the Large Hadron Collider (LHC) at the European Organization for Nuclear Research CERN will result in never before reached luminosities and very high pile-up conditions and will thus impose new challenges on the experimental setups of the four big detectors located at the LHC.

In order to maintain or even improve upon the current performance capabilities under these new challenging operating conditions, and hence guarantee a full exploitation of the rich physics potential provided by the LHC, also major upgrades of the ATLAS sub-detector systems are required.

The ATLAS detector observes proton-proton collision events at the LHC bunch crossing rate of 40 MHz. In order to reduce the data flow to manageable levels and keep only the small fraction of events containing physics processes that are of interest for further analyses, efficient online data selection of interesting and rare processes out of the large data volume delivered by the LHC is essential for a collider experiment. The ATLAS trigger system is specifically designed for this important task. It consists of a hardware-based Level-1 trigger and a software-based High-Level trigger, reducing the event rate from the LHC bunch crossing rate to the detector read-out rate of about 1 kHz.

Among others, the Level-1 trigger system involves the Level-1 Calorimeter Trigger (L1Calo) which is specialized on searches for e.g. high-energy electrons, photons, τ leptons and particle jets. Jets play crucial roles in many analyses like e.g. searches for new physics and are therefore important objects that need to be reconstructed in the trigger system. An efficient jet finding algorithm implemented on FPGAs in L1Calo is indispensable to cope with the upcoming very busy environments.

By considering calorimeter information as images, state-of-the-art image recognition techniques like convolutional neural networks become promising candidates for this challenging task. In the last decade, the development of deep learning and image recognition techniques has made an enormous progress, often leading to novel algorithms that are able to outperform previous state-of-the-art techniques. Moreover, the Universal Approximation Theorem [1] suggests that neural networks can be regarded as universal approximators for a variety of different problems. This motivates the consideration of utilizing a deep artificial neural network for jet reconstruction in the

ATLAS Level-1 Calorimeter Trigger.

The development of a neural network that is capable of finding jets with an efficiency of the anti- k_t jet clustering algorithm while simultaneously being small enough to allow an implementation on the Level-1 trigger hardware is the goal of this thesis and will be presented in the following.

Chapter 2

The Standard Model of Particle Physics

The Standard Model (SM) of particle physics is a theory that describes the fundamental interactions between particles with the exception of gravity. It has been remarkably successful in describing a wide range of particle physics processes as well as in predicting fundamental particles, all of which have been found, the last one in 2012 at the Large Hardron Collider at CERN.

2.1 Fundamental Interactions and Particle Content

The Standard Model is a renormalizable quantum field theory, based on an underlying $SU_C(3) \times SU_L(2) \times U_Y(1)$ symmetry group [2].

Here, $SU_C(3)$ is the gauge group of quantum chromodynamics (QCD), the theory of strong interaction. Its eight generators $G^\alpha, \alpha = 1, \dots, 8$ corresponds to the massless spin-1 force mediators of QCD, the *gluons*. Particles that are charged under $SU_C(3)$ are said to carry *color*, hence the subscript “C”. The non-abelian character of QCD leads to the force carriers themselves carrying color charge, resulting in gluon self-interaction.

$SU_L(2) \times U_Y(1)$ represents the electroweak sector, where the subscript “L” indicates that the $SU_L(2)$ symmetry group acts exclusively on left-handed particles (and right-handed antiparticles) and the subscript “Y” denotes the *weak hypercharge* Y. The four generators of the electroweak symmetry group lead to four massless gauge bosons, which are denoted as $W^a, a = 1, \dots, 3$ for $SU_L(2)$ and B for $U_Y(1)$.

The massive $W^\pm$ and Z^0 bosons, the charged and neutral force carriers of the weak interaction, as well as the *photon* γ , mediator of the electromagnetic force, are produced by *spontaneous symmetry breaking* of the electroweak symmetry, caused by the *Higgs mechanism*: The W^3 and B bosons mix to form the Z^0 boson and the photon whereas W^1 and W^2 combine to give the charged bosons $W^\pm$.

Apart from the *gauge bosons* described above, the SM includes twelve spin-1/2 particles, called *fermions*. These are grouped into three generations, as illustrated in Fig. 2.1 and can be divided into two different types of fermions, *quarks* and *leptons*.

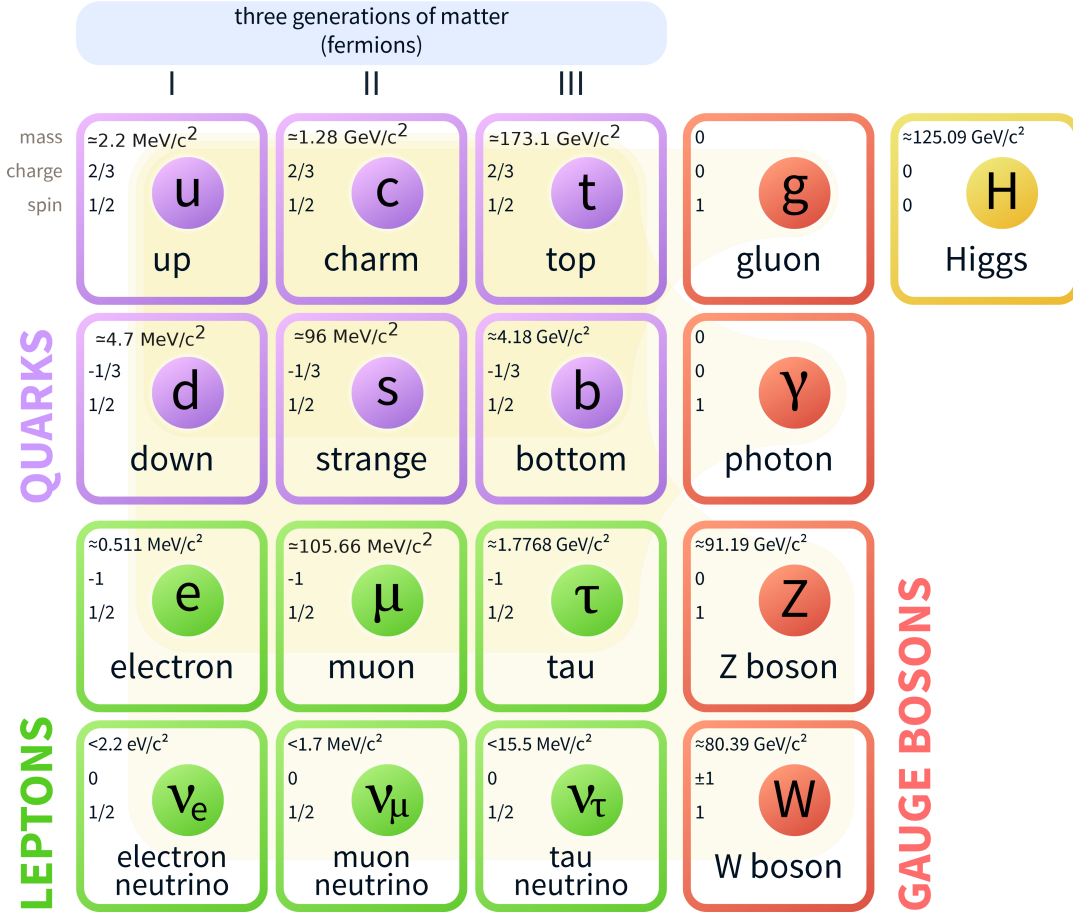


Figure 2.1: Summary of elementary particles in the Standard Model [3].

There are six quark flavors, called *up*(*u*), *down*(*d*), *charm*(*c*), *strange*(*s*), *top*(*t*) and *bottom*(*b*). They all have fractional electric charge values. Up-, charm- and top-quarks (collectively referred to as up-type quarks) have an electric charge of $+\frac{2}{3}e$, while down-, strange- and bottom-quarks (down-type quarks) have $-\frac{1}{3}e$. In addition, quarks carry color charge, which makes them — besides interacting via the electromagnetic and weak force — also interact strongly. Particles that carry color charge cannot exist isolated but only in colorless bound states (called *hadrons*). This important property of QCD is called *confinement* and is discussed in more detail in Section 2.2.2.

Leptons, on the other hand, do not carry color charge and thus do not take part in the strong interaction. There are three electrical charged lepton flavors (*electron*, *muon*, *tau*) with charge $-1e$ and three corresponding neutrino flavors (*electron neutrino*, *muon neutrino*, *tau neutrino*), which do not carry electric charge and therefore only interact weakly. Each of the twelve fermions comes with a corresponding antiparticle with the same properties except of opposite charges.

All fermions as well as the massive gauge bosons acquire their masses by interacting with a scalar field, called the Higgs field, introduced by the Higgs mechanism. This field comes along with a scalar spin-0 particle, the *Higgs boson*, which completes the list of known elementary particles in the SM.

2.2 Quantum Chromodynamics at Hadron Colliders

At hadron colliders like the Large Hadron Collider at CERN (see Chapter 3.1), QCD processes are omnipresent and hence play an important role in data analysis, such as precision studies of the SM or searches for new physics. This chapter briefly covers the most relevant concepts of QCD for the context of this thesis.

2.2.1 The QCD Lagrangian

The QCD part of the Standard Model Lagrangian reads

$$\mathcal{L}_{QCD} = -\frac{1}{4}F^{a\mu\nu}F_{\mu\nu}^a + i\bar{\psi}_i\gamma^\mu D_\mu\psi_i \quad (2.1)$$

with the field strength tensor

$$F_{\mu\nu}^a = \partial_\mu A_\nu^a - \partial_\nu A_\mu^a + g_s f^{abc} A_\mu^b A_\nu^c. \quad (2.2)$$

Here, A_μ^a are the gluon fields, g_s the coupling constant, f_{abc} the structure constant and

$$D_\mu = \partial_\mu - ig_s t^a A_\mu^a \quad (2.3)$$

the covariant derivative, where t^a are the generating matrices of $SU_C(3)$.

The last term in Eq. (2.1) together with Eq. (2.3) gives rise to the

- Quark-gluon vertex $-g_s\bar{\psi}t^a\gamma_\mu A^{a\mu}\psi$,

whereas the last term in Eq. (2.2) together with the first term of Eq. (2.1) is responsible for the

- Gluon-gluon vertices $g_s f^{abc}(\partial^\mu A^{a\nu})A_\mu^b A_\nu^c$ and $g_s^2 f^{abc} f^{ade} A^{b\mu} A^{c\nu} A_\mu^d A_\nu^e$.

This gluon self-interaction results in interesting properties of QCD which will be briefly discussed in the following.

2.2.2 Asymptotic Freedom and Confinement

In quantum electrodynamics (QED), a charged particle is surrounded by short-lived electron-positron pairs that are continuously created from the vacuum and annihilate again within a time window allowed by the Heisenberg uncertainty relation

$\Delta E \Delta t > \hbar/2$. Since opposite charges attract each other, the charged particle polarizes the electron-positron pairs, resulting in a phenomenon called *vacuum polarization*. Vacuum polarization makes the vacuum a dielectric medium which leads to partial **screening** of the electric charge. At high energies, i.e. small distances, the screening becomes less influential and the effective charge $e(r)$ increases. Analogously, the effective charge becomes smaller for large distances.

One says that the QED beta function

$$\beta(r) \equiv -\frac{de(r)}{d \ln r} \quad (2.4)$$

is positive.

Likewise, the QCD vacuum contains virtual quark-antiquark pairs that would also result in a positive beta function. However, gluon self-interaction allows virtual gluon pairs to be created from the vacuum. Since gluons carry color charge (and anti-color charge) themselves, the gluon cloud surrounding the bare charge results in an effective charge that becomes smaller with smaller distances.

This effect is called **antiscreening**. The latter effect is stronger than the former one, resulting in an overall negative QCD beta function. QCD charge antiscreening motivates the concept of the *running coupling constant*. The strong coupling constant becomes smaller at smaller distances (larger momentum transfers), leading to quarks behaving more and more freely when being probed at larger energies. This is referred to as **asymptotic freedom** and allows perturbative QCD (pQCD) approaches for hard scattering processes at hadron colliders.

On the other hand, the coupling increases more and more for larger distances. An isolation of a single quark is therefore not possible, in fact it is energetically more favorable to produce new quark-antiquark pairs from the vacuum to create bound states. This phenomenon is called **color confinement** and leads to the existence of only colorless hadrons, meaning that colored particles cannot exist isolated.

2.2.3 Hadronic Jet Production

In QCD hard scattering processes, high transverse momentum quarks and gluons (collectively called *partons*) are produced, which result in a collimated stream or **jet** of particles, inheriting the initial parton's energy and momentum. Jets are important physics objects in hadron collider experiments as they are easy to measure in the detector and play a crucial role in e.g. the search for new physics.

Jet production occurs in three consecutive steps: *hard process*, *parton shower* and *hadronization*.

Hard Scattering Process

Scattering processes with large momentum transfers Q are called hard scattering processes. The cross section for a hard scattering process of two incoming hadrons with momenta P_1 and P_2 can be expressed in terms of the parton level cross section $\hat{\sigma}_{ij}$ by

$$\sigma(P_1, P_2) = \sum_{i,j} \int dx_1 dx_2 f_i(x_1, \mu_f) f_j(x_2, \mu_f) \hat{\sigma}_{ij}(x_1 P_1, x_2 P_2, Q/\mu_f). \quad (2.5)$$

Here, the sum extends over all initial-state partons $i, j \in \{q, \bar{q}, g\}$ and $f_i(x_k, \mu_f)$ denotes the *parton distribution function* (PDF), giving the probability to find a parton of flavor i carrying momentum fraction x_k of the incoming hadron $k \in \{1, 2\}$ at the factorization scale μ_f .

Partonic Shower Production

Partons in the final state have a high probability of radiating extra partons. Quarks, antiquarks and gluons can radiate an additional gluon, gluons can split into quark-antiquark pairs. This kind of extra radiation is particularly strong when the radiated parton is collinear with the original one (collinear divergence) or very soft (infrared divergence). These two types of radiative enhancements combine to the concept of a *parton shower*.

Hadronization

Due to color confinement the shower partons cannot exist individually and start forming color-neutral hadrons by creating quark-antiquark pairs from the vacuum to combine with. This process is called *hadronization*.

The theory of hadronization is not yet fully understood but can be described using phenomenological models. The most common one is the *Lund string model* [4], schematically illustrated in Fig. 2.2. It is based on the idea of a linearly increasing

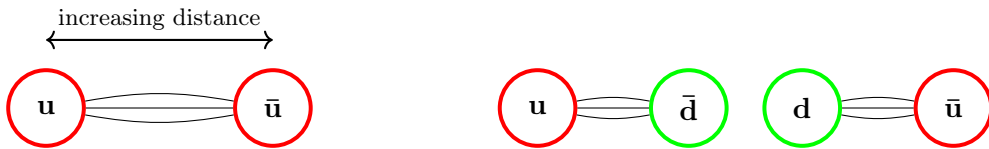


Figure 2.2: Hadronization in the string model. The potential energy increases as the quarks move further apart (left). Eventually the string breaks, forming a new quark-antiquark pair (right).

potential with the distance between two quarks, whose energy is stored in strings connecting the quarks. At a certain distance, the potential energy is so large that the string breaks and a quark-antiquark pair is created from the vacuum.

The monte carlo event generator PYTHIA [5], utilized for data production in this thesis

(see Section 5), calculates jet developments based on this string model.

The consecutive sequence of hard scattering and parton showers, followed by hadronization leads to a narrow cone of collimated particles, the jet. The production of three jets in a proton-proton scattering process at parton, hadron and detector-level as well as the associated typical length scales are depicted in Fig. 2.3.

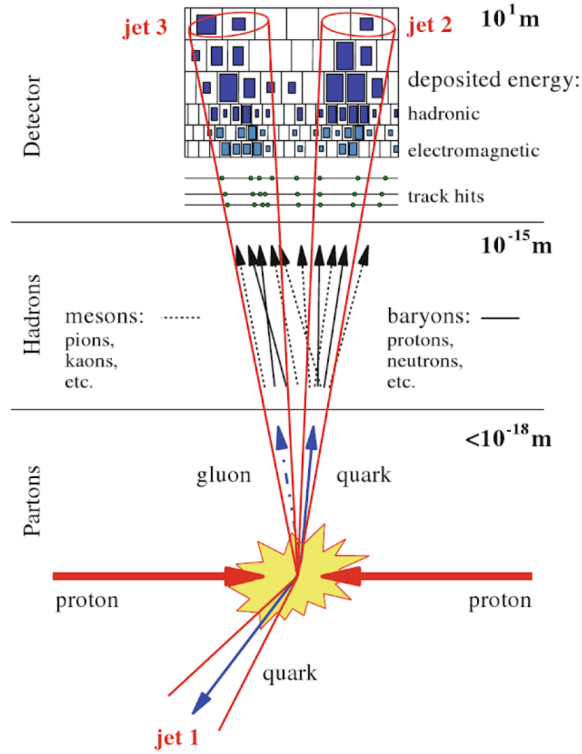


Figure 2.3: Illustration of jet production at parton, hadron and detector-level [6].

2.3 Beyond the Standard Model

Despite being extremely successful in describing a wide range of particle physics processes, the standard model is not perfect. There are fundamental physical phenomena and observations that the SM is not able to explain, such as gravity, matter-antimatter asymmetry, neutrino masses or dark energy and dark matter.

Searches for physics beyond the SM are one of the main research topics at the present time. These include amongst many others for example searches for mono-jet events at the experiments at the LHC at CERN, as they could indicate the existence of dark matter.

Chapter 3

The ATLAS Experiment

3.1 Experiments at the Large Hadron Collider

The Large Hadron Collider is the world's largest and most powerful particle accelerator. It is a 27 km proton-proton storage ring and part of CERN's particle accelerator complex as shown in Fig. 3.1. Two proton beams — and in special runs also beams of other particles such as lead ions — are being accelerated in the LHC to almost the speed of light as they travel in opposite directions in separate beam pipes. In normal operation, each beam consists of about 2800 bunches of about 10^{11} protons per bunch. The accelerator is designed to provide proton-proton collisions at a center-of-mass (CM) energy of 14 TeV with a design luminosity of $10^{34} \text{ cm}^{-2} \text{ s}^{-1}$. This value was already doubled in 2017. The particle bunches are separated by 25 ns, resulting in a collision rate of 40 MHz in the experiments located at the LHC. There are four main experiments at the LHC, whose main physics goals will be shortly described in the following:

- **ATLAS** is one of two large multi-purpose detectors at the LHC, covering a wide range of particle physics research and designed to exploit the full discovery potential of the LHC. It is optimized for the discovery and precise measurement of the Higgs boson as well as for SM precision tests, searches beyond the SM such as searches for dark matter and also heavy-ion physics.
- **ALICE** is a heavy-ion detector, dedicated to the study of strongly interacting matter at extremely high densities, where the existence of quark-gluon plasma is expected.
- **CMS** is the second multi-purpose detector at the LHC, following a similar physics program like ATLAS.
- **LHCb** is specialized on b-physics and precision studies of CP violation and thus aiming to provide for example insight into the phenomenon of matter-antimatter asymmetry.

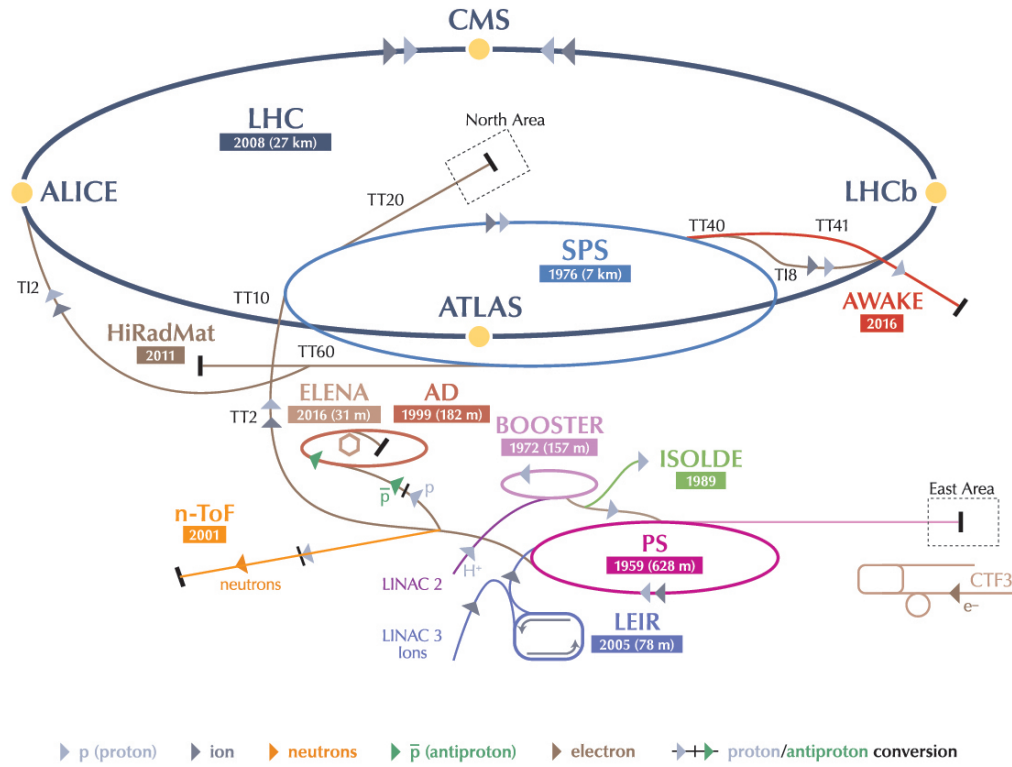


Figure 3.1: The CERN accelerator complex. The LHC is the last stage in a chain of accelerators, where the smaller machines upstream of the LHC are used to boost the particles to their final energies as well as to provide particle beams to several other smaller experiments [7].

3.2 ATLAS Coordinate System and Definitions

The ATLAS coordinate system is a right-handed coordinate system centered at the nominal collision point. The z -axis points along the beam pipe, the positive x -axis to the center of the LHC ring and the positive y -axis upwards to the surface of the Earth. The x - y plane is referred to as the “transverse plane”. In order to denote coordinates, the azimuth angle ϕ , measured from the positive x -axis in the transverse plane, as well as the polar angle θ , defined as the angle from the positive z -axis in the y - z plane, are used.

The polar angle is often given in terms of a massless approximation of the Lorentz-invariant rapidity, known as *pseudo-rapidity* η . It is given by

$$\eta = \ln \frac{|\vec{p}| + p_L}{|\vec{p}| - p_L} = -\ln \tan \frac{\theta}{2}, \quad (3.1)$$

where $\vec{p}$ and p_L are the momentum and longitudinal momentum of a particle respectively.

In order to quantify distances between various objects in the detector, the quantity $\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$ is defined.

The transverse momentum p_T is given by $p_T = \sqrt{p_x^2 + p_y^2}$, the transverse energy E_T is

defined accordingly as $E_T = \sqrt{m^2 + p_T^2}$ for a particle with mass m ($c = 1$). These are important quantities as they can be used to identify particles that left the detector without being detected: Since the incoming partons have no transverse momentum, the total sum of transverse momenta of all produced particles has to be zero as well due to momentum conservation. Values different from zero imply undetected particles.

3.3 Detector Components

ATLAS is a cylindrical symmetric detector with 46 m in length and 25 m in diameter, making it the largest volume particle detector ever built for a collider. It features an onion-like structure, where the various sub-detector components are specialized on precise measurements of different properties of the particles emerging from the interaction point. The innermost detector layer is a tracking system surrounding the interaction point. It is followed by the electromagnetic and hadronic calorimeters, which are in turn enclosed by the muon system. A schematic overview, also including the toroid and solenoid magnets, is shown in Fig. 3.2.

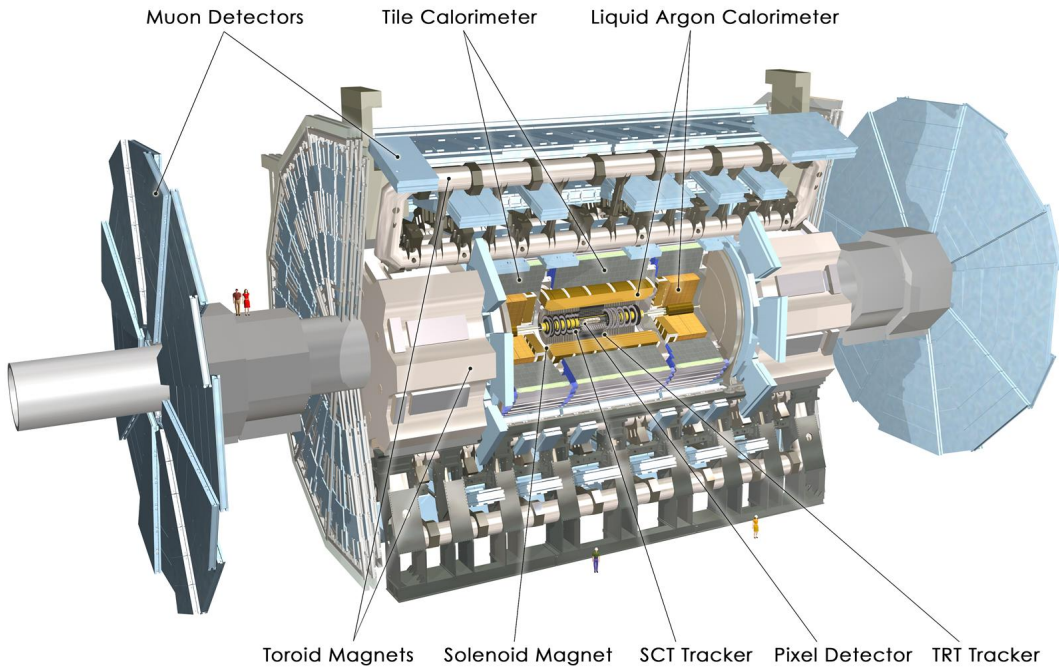


Figure 3.2: The ATLAS detector at CERN with its sub-detector and magnet systems [8].

A brief description of the detector components will be given in the following.

3.3.1 Inner Detector and Central Solenoid

The inner detector [9] is the innermost layer and begins 3.3 cm from the beam axis and extends up to a radius of 1.1 m. The acceptance region in pseudo-rapidity for

particles coming from the beam interaction point is $|\eta| < 2.5$ with a full 2π -coverage in ϕ .

It detects interactions of electrically-charged particles at discrete points and is thus able to track the particle's trajectory. The central solenoid magnet in which the inner detector is fully contained causes the particle's path to curve, allowing to gain information about the momentum and charge of the particle. It produces a strong 2 T magnetic field whose axis coincides with the beam axis.

The inner detector comprises three sub-detector components: the pixel detector, the semi-conductor tracker and the transition radiation tracker.

The pixel detector, the innermost part of the inner detector, consists of 1744 modules of roughly 47000 silicon pixels per module. Together with the Insertable B-Layer, the innermost layer of the pixel detector which was installed in 2014, it results in more than 180 million readout channels, which is more than 50% of the total readout channels of the whole detector.

The semi-conductor tracker, the middle component of the inner detector, features a similar concept as the pixel detector, but utilizes long and narrow strips rather than small pixels in order to make the coverage of the larger area more practical.

The outermost component, the transition radiation tracker which uses proportional drift tubes as opposed to semi-conductors, completes the inner detector.

3.3.2 Calorimeters

One very important component of a detector is the calorimeter system [10, 11], which is dedicated to measure the particle energies as they traverse through the detector. The dense material causes the particles to produce particle showers, which then deposit their energy in the calorimeter cells. The ATLAS calorimeter, shown in Fig. 3.3, consists of an electromagnetic and a hadronic calorimeter, each of which built such that electromagnetic and hadronic showers are fully contained in the respective calorimeter parts in order to optimize the energy resolution.

The relative energy resolution of a calorimeter is given by

$$\left(\frac{\Delta E}{E}\right)^2 = \left(\frac{c_s}{\sqrt{E}}\right)^2 + \left(\frac{c_n}{E}\right)^2 + \left(c_c\right)^2, \quad (3.2)$$

with coefficient c_s , c_n and c_c , describing stochastic, noise-induced and a constant contribution respectively. Given that the number of shower particles increases with energy, the stochastic term is proportional to $1/\sqrt{E}$, whereas detector noise is not dependent on the deposited energy and is therefore generally negligible at very high energies. The constant term mainly accounts for shower tails that are not captured by the detector.

Apart from being able to perform precise energy measurements, the calorimeters also serve as input for the ATLAS trigger system (see Section 3.4). They provide fast signal responses which are then — among others — used to decide if an event is interesting

enough to be stored for further analysis.

Since the ATLAS calorimeter system plays an important role in the context of this thesis, the design and functionality of the electromagnetic and hadronic calorimeter will be discussed separately in more detail in the following.

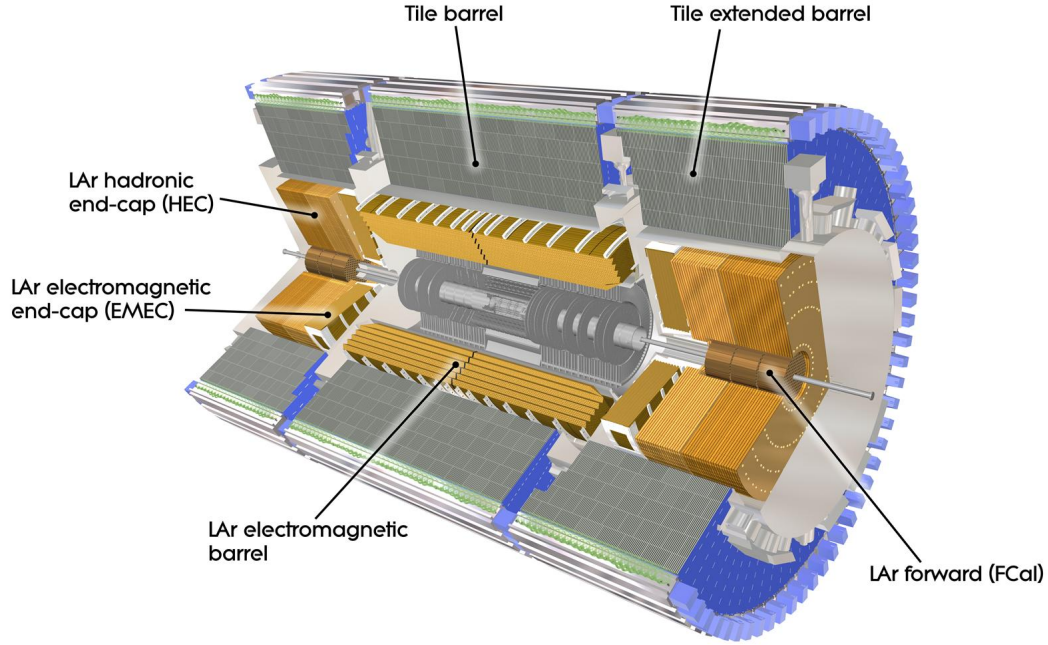


Figure 3.3: Schematic view of the ATLAS calorimeter system [12].

Electromagnetic Calorimeter

The electromagnetic calorimeter (ECAL) is designed to measure electromagnetic showers produced by electrons and photons. It is built as a sampling detector, i.e. it consists of alternating layers of absorber and active material. In ATLAS, liquid argon (LAr) is used as an active material, combined with lead (Pb) as the absorber and an accordion-like geometry is employed, as illustrated in Fig. 3.4(a). The ECAL provides a full azimuth coverage and is divided into a barrel part and two end-caps. The barrel section starts at a radius of 115 cm and extends up to a radius of 225 cm, covering a pseudo-rapidity range of $|\eta| < 1.475$. The end-caps cover a region between $1.375 < |\eta| < 3.2$.

The amount of deposited energy is proportional to the LAr ionization signal, which is read out using electrodes in up to three longitudinal sampling layers with varying granularity between (0.003×0.1) and (0.05×0.025) in $(\Delta\eta \times \Delta\phi)$, as depicted in Fig. 3.4(a). *Trigger towers*, a summation of electromagnetic and hadronic calorimeter cells in width and depth that serve as input for the calorimeter trigger, come with a coarser granularity: The current system in Run-2 features a granularity of $\Delta\eta = 0.2$ and $\Delta\phi = 2\pi/32 \approx 0.1963 \approx 0.2$ while the granularity will be $\Delta\eta = 0.1$ and

$\Delta\phi = 2\pi/64 \approx 0.0981 \approx 0.1$ in Run-3.

A total depth of more than 20 radiation lengths (X_0) ensures a full shower containment in the calorimeter [13].

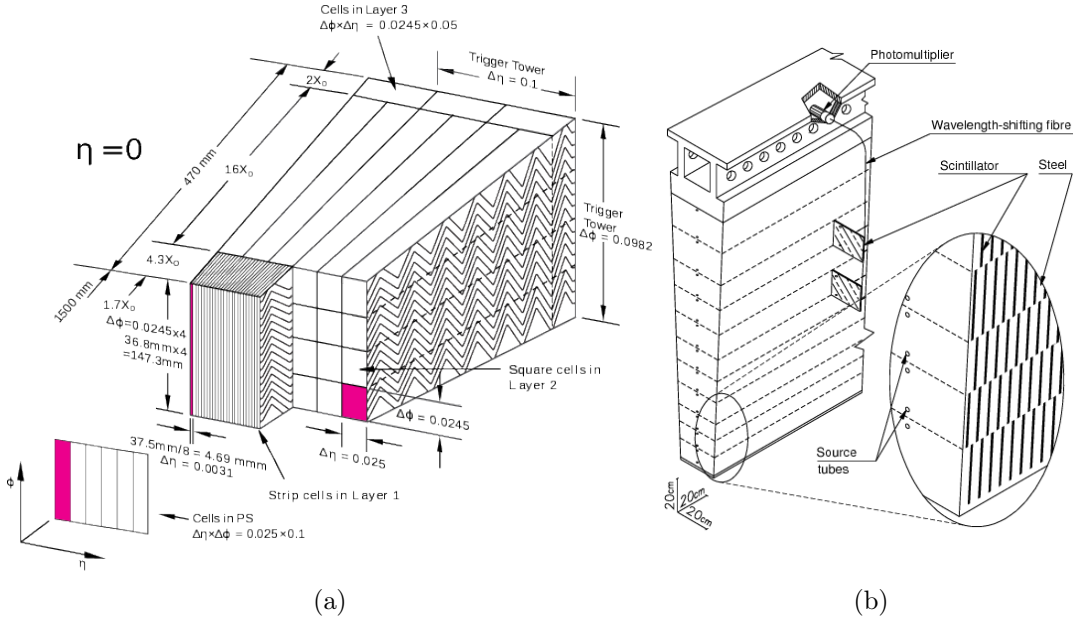


Figure 3.4: Schematic view of (a) one ECAL barrel module at $\eta = 0$ with its accordion structure and the subdivision into three longitudinal layers and (b) one hadronic tile calorimeter module, showing the alternating scintillator and steel layers [14, 15].

Hadronic Calorimeter

Energies of hadrons produced in quark- or gluon-initiated particle jets, decays of τ leptons or other hadron decays are measured in the hadronic calorimeter (HCAL). Like the ECAL, the HCAL is built as a sampling detector and comprises two different calorimeter systems: The barrel tile calorimeter combines steel and scintillators as detector materials and covers a region of $|\eta| < 1.7$. It has a radial extent between 228 cm and 423 cm and is complemented by the hadronic end-cap calorimeter. The hadronic end-cap uses LAr in combination with copper, covering $1.5 < |\eta| < 3.2$. Each cell has a transversal granularity of (0.1×0.1) in $(\Delta\eta \times \Delta\phi)$.

At $\eta = 0$, the total depth of the hadronic calorimeter amounts to 7.4 nuclear interaction lengths (λ), 9.7 λ when including the LAr ECAL and tracking system [13].

A schematic view of a barrel tile calorimeter module is given in Fig. 3.4(b).

Forward Calorimeter

The forward calorimeter (FCAL) [16] covers a pseudo-rapidity range of $3.1 < |\eta| < 4.9$ and combines hadronic and electromagnetic calorimetry in one system. It consists of three separate layers, where the first one (FCAL 1) is optimized for measurements

of electromagnetic, the second and third layer (FCAL 2 and FCAL 3) for hadronic interactions. LAr is used as the active detector material and copper (Cu) or tungsten (W) as the passive absorber.

The following table summarizes the main properties of the ATLAS calorimeter system [6], including the energy resolutions of the respective subsystems in terms of the coefficients c_s and c_c as introduced in Eq. (3.2).

Calorimeter		Materials	Angular coverage	Energy resolution	
				$c_s(\%)$	$c_c(\%)$
ECAL	barrel	LAr + Pb	$ \eta < 1.475$	10	0.7
	end-cap	LAr + Pb	$1.375 < \eta < 3.2$	10	0.7
	FCAL	LAr + Cu	$3.1 < \eta < 4.9$	28.5	3.5
HCAL	barrel	Scint. + steel	$ \eta < 1.7$	52	3
	end-cap	LAr + Cu	$1.5 < \eta < 3.2$	84	-
	FCAL	LAr + W	$3.1 < \eta < 4.9$	94	7.5

Table 3.1: Overview of the main properties of the ATLAS calorimeters.

3.3.3 Muon Spectrometer and Toroid Magnet System

The muon spectrometer [17] is the outermost component of the ATLAS detector and extends from a radius of 4.25 m up to 11 m, the full radius of the detector. It is composed of two gas-based sub-detector systems: precision chambers for good momentum resolution measurements and trigger chambers, which feature fast response times for a high trigger efficiency.

The precision muon tracking system consists of monitored drift tubes, covering a pseudo-rapidity range of $|\eta| < 2.0$, and cathode-strip chambers for $2.0 < |\eta| < 2.7$. The muon trigger chambers are also subdivided into two separate systems: resistive-plate chambers with a coverage of $|\eta| < 1.05$ and thin-gap chambers, which cover a range of $1.05 < |\eta| < 2.4$.

The toroid magnet system, which is situated outside the calorimeter and within the muon system, consists of eight barrel coils and two end-caps. It creates a magnetic field of up to 4 T.

3.4 Trigger System

With the LHC delivering bunch crossings at a rate of about 40 MHz and ATLAS only being capable of writing data to permanent storage at a rate of about 1 kHz, the trigger system becomes an essential component of the detector. Furthermore, most of

the interesting physics processes are very rare, having cross sections several orders of magnitude smaller than the inelastic proton-proton scattering. In order to keep the interesting events, a trigger system that makes decisions on an event-by-event basis whether or not an event is interesting enough to be recorded for further analysis is indispensable.

3.4.1 Design and Functionality

Fig. 3.5 shows the current ATLAS trigger system [18] as it is employed in Run-2 (2015-2018). It consists of two stages: a hardware-based first level trigger (Level-1) and a software-based high level trigger (HLT). The Level-1 trigger, which reduces

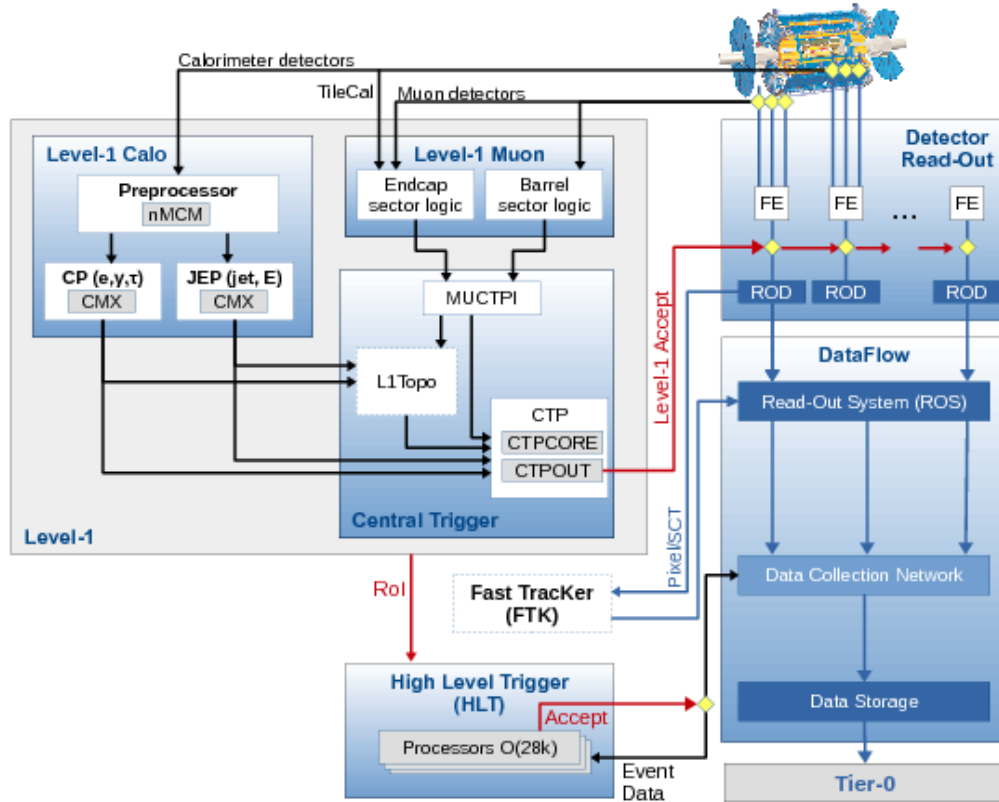


Figure 3.5: The ATLAS Trigger and Data Acquisition (TDAQ) system in Run-2 [18].

the event rate from 40 MHz to about 100 kHz, receives coarse granularity calorimeter information as well as muon detector information as input and has a decision time for a Level-1 accept of $2.5 \mu\text{s}$. The Level-1 accept decision is formed by the Central Trigger Processor (CTP) based on information received from the Level-1 Calorimeter Trigger (L1Calo), Level-1 Muon Trigger (L1Muon) and the topological trigger processor (L1Topo). The scope of L1Calo comprises searches for high transverse-energy electrons, photons, jets and τ leptons as well as high missing and total transverse energy, whereas L1Muon is dedicated to muons.

L1Topo, consisting of two FPGA-based (Field-Programmable Gate Arrays) processor modules, takes trigger objects from the Level-1 Calorimeter Trigger and data from MuCTPi (Muon-to-Central-Trigger-Processor Interface) as input, allowing it to apply topological selections, such as invariant mass requirements or angular separations. So-called Regions of Interest (RoIs) are also determined in the Level-1 trigger which are passed to the HLT. The HLT applies more advanced algorithms on either RoIs or the entire event using the full available granularity detector information. In this second trigger stage, the event rate is further reduced to the detector read-out rate of about 1 kHz.

Sliding Window Jet Finding Algorithm

Finding jet candidates is one of the main tasks of the Level-1 Calorimeter Trigger. Since this thesis aims to improve the jet finding in the Level-1 Calorimeter Trigger by making use of deep learning methods, the algorithm that is currently being employed in Run-2 will be briefly explained in the following, based on [19].

A 2×2 sum of trigger towers, giving a granularity of 0.2×0.2 in $\Delta\eta \times \Delta\phi$, forms the basic unit for this algorithm, called a *jet element*, where electromagnetic and hadronic calorimeter are summed in depth. The algorithm has two key components:

- Jet RoIs, defined as 2×2 jet element windows, used to identify jet candidates.
- Jet windows with sizes 0.4×0.4 , 0.6×0.6 and 0.8×0.8 in $\Delta\eta \times \Delta\phi$ (see Fig. 3.6), which are used to measure the jet E_T .

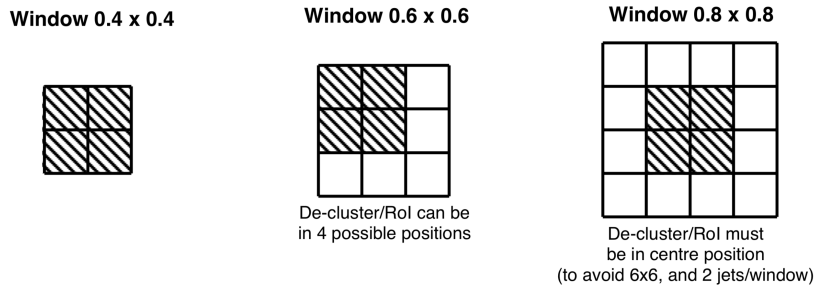


Figure 3.6: Window sizes used for the sliding window jet algorithm based on 0.2×0.2 jet elements. RoIs are represented as shaded areas [19].

There are two requirements to be fulfilled in order to consider a certain region to contain a jet trigger candidate. First, the RoI cluster must be a local E_T maximum, compared to its neighbors. In addition, the jet window E_T sum must exceed a certain predefined trigger threshold. In the case of the 0.6×0.6 window, where four windows containing a given RoI are possible, the one with the highest E_T sum is used. The jet windows slide across the entire calorimeter (hence this algorithm is often referred to as the “sliding window algorithm”) in steps of jet elements in both η and ϕ direction searching for jet candidates.

3.4.2 Phase-1 Upgrade of First Level Calorimeter Trigger

The ATLAS trigger system will face new challenges in the upcoming years as the instantaneous luminosity delivered by the LHC will even further increase in Run-3 (2020-2022) and beyond. In order to cope with an expected luminosity of more than $3 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$ an upgrade of the ATLAS trigger system in the context of the intended Phase-1 upgrade (2019-2020) is mandatory.

In addition to the current system, three new subsystems, called *feature extractors* (FEX), will be added to L1Calo as depicted in Fig. 3.7. The Electron Feature Extractor eFEX, Jet Feature Extractor jFEX and the Global Feature Extractor gFEX will eventually replace the previous system.

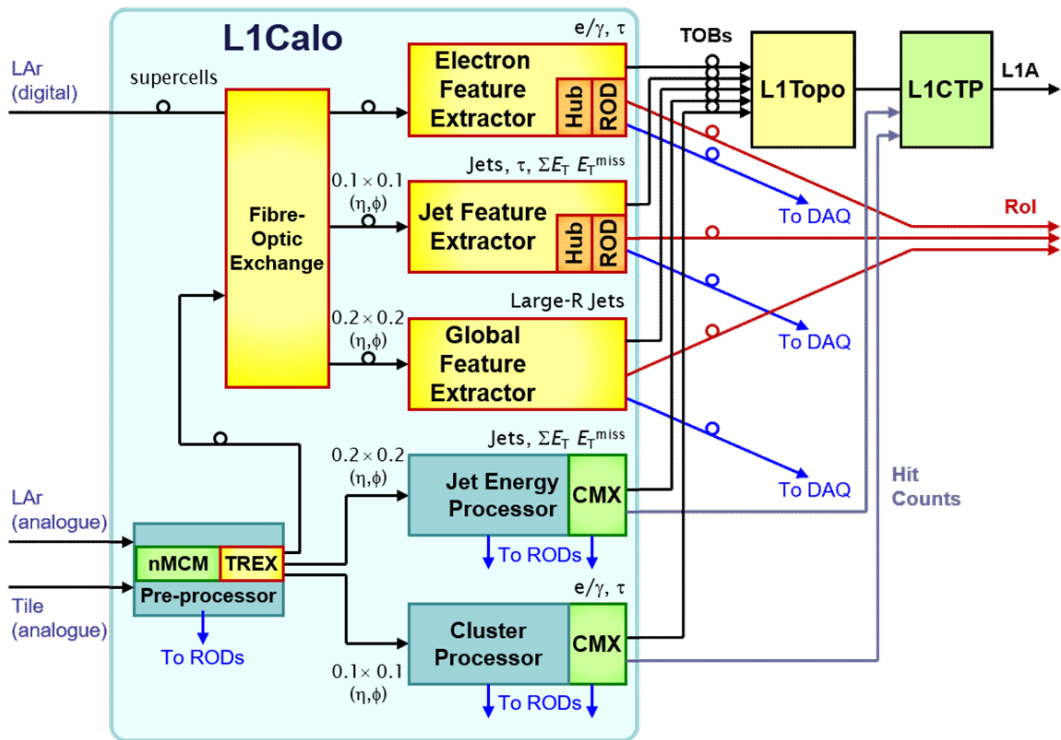


Figure 3.7: Schematic view of the new L1Calo system for Run-3. The green components show the current system, which will be supplemented and eventually replaced by the yellow components as part of the Phase-1 Upgrade [20].

The eFEX subsystem is designed to identify isolated electron, photons and τ leptons, whereas the main scope of gFEX will be the identification of large-area jets for capturing boosted objects such as top quarks, Higgs, Z or W bosons.

jFEX plays an important role in the context of this thesis since the jet reconstruction in the Level-1 trigger will be performed on this subsystem. Hence, the most relevant aspects and functionality of jFEX will be briefly discussed in the following.

Jet Feature Extractor (jFEX)

jFEX receives input data from the central and forward calorimeters with a finer granularity (0.1×0.1 in $\eta \times \phi$) compared to its predecessor (Jet Energy Processor with 0.2×0.2 in $\eta \times \phi$). The real-time identification of jets and large-area τ candidates as well as the calculation of the total transverse energy E_T and missing transverse energy E_T^{miss} are the tasks jFEX has to accomplish.

The entire calorimeter is covered using six jFEX modules, four modules for the central region and two for the forward region, each comprising four processor FPGAs.

Fig. 3.8(a) illustrates the calorimeter subregion as seen by one single FPGA in the central region. The blue region is the FPGA core region in which the jet finding algorithm on this particular FPGA is supposed to find jets. The green area indicates the overlap (“environment”) with the neighboring FPGAs. With a single FPGA core region providing a ϕ -coverage of $\pi/2 \approx 1.6$, a jFEX module covers exactly one ϕ -ring ($4 \times \pi/2 = 2\pi$) of the calorimeter.

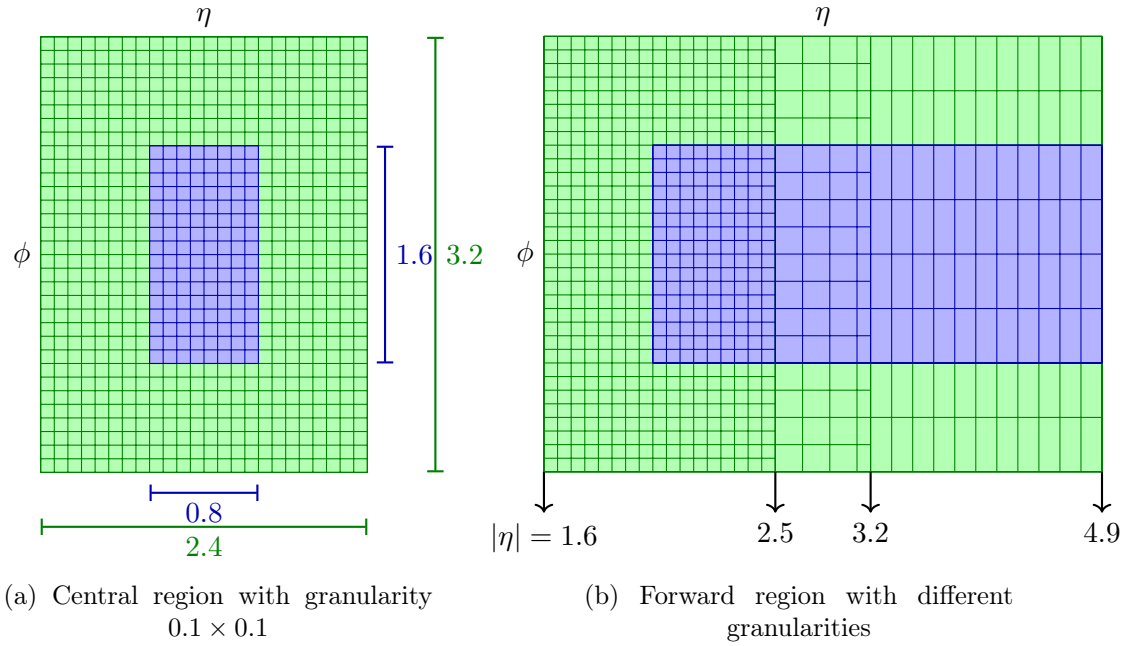


Figure 3.8: FPGA core (blue) and environment (green) in jFEX for central and forward region.

The calorimeter subregion as seen by one FPGA in the forward region is depicted in Fig. 3.8(b). Here, the input granularity varies for different η values and becomes more coarse for $\eta > 2.5$.

3.5 Jet Reconstruction at ATLAS

A jet algorithm that is applicable to theoretical calculations as well as experimental data is an important tool in a collider experiment as one wants to compare theoretical predictions with measured quantities in the detector. Collinear and soft radiation does not influence the jet development, what has to be reflected in a jet algorithm. Hence, it has to be collinear- and infrared safe. Collinear safety means that the result of the clustering does not change due to merging or splitting of collinear parton four-vectors, infrared safety refers to an unchanged outcome of the clustering when arbitrarily soft partons are added to the event. In addition, a jet algorithm has to be computational efficient and should not depend on detector details.

One class of jet algorithms that fulfill these requirements are sequential-recombination algorithms that use certain distance measures to determine closest objects and combine them iteratively.

In ATLAS, one specific algorithm of this class, the so called anti- k_t algorithm, is the method of choice. It is discussed in the following.

3.5.1 The Anti- k_t Jet Clustering Algorithm

The distance measures used by the anti- k_t algorithm [21] are defined as

$$d_{ij} = \min(k_{ti}^{2p}, k_{tj}^{2p}) \frac{\Delta_{ij}^2}{R^2}, \quad (3.3a)$$

$$d_{iB} = k_{ti}^{2p}, \quad (3.3b)$$

where $\Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$ and k_{ti} , y_i and ϕ_i are the transverse momentum, rapidity and azimuth of particle i respectively. R describes the radius parameter and p governs the relative power between the energy and distance scale Δ_{ij} . The anti- k_t algorithm is defined by $p = -1$.

The algorithm identifies the smallest of the distances d_{ij} and d_{iB} , where d_{ij} is the distance between two entities i and j (particles or pseudojets) and d_{iB} the distance between entity i and the beam. If d_{ij} is the smallest distance, i and j are replaced by a new single object with momentum $p_i + p_j$. Since this object is neither a single particle nor a full jet yet, it is often called a “pseudojet”. If, however, d_{iB} is the smallest distance, i is called a jet and is removed from the event list. The algorithm continues until no particles or pseudojets remain.

In ATLAS, the choice of the radius parameter is $R = 0.4$ and $R = 1.0$ for large-area jets.

Chapter 4

Deep Learning Methods

Deep learning, a subset of machine learning, has achieved tremendous success in the recent years. Many previous state-of-the-art algorithms in various fields were outperformed by deep learning methods, often reaching superhuman abilities.

Deep learning methods make use of specific algorithms, called *Deep Neural Networks*. In the case of so-called supervised learning, which is what we will focus on, the network is trained by showing it lots of examples to learn from. After the network has been trained sufficiently, one can use it to make prediction on data it has never seen before. This chapter gives an overview over all important concepts of deep neural networks that will be relevant in this thesis, based on [22].

4.1 Deep Feed Forward Neural Networks

4.1.1 The Perceptron

The most simple form of a neural network is the so-called *perceptron*, a single-layer neural network which consists of only one *neuron*. A neuron is the fundamental unit of a neural network and is schematically shown in Fig. 4.1. Neurons are mathematical functions that receive one or several inputs and produce a correspondent output: The neuron multiplies each input x_i with a weighting factor w_i and sums them up. After adding a bias b , the sum is passed to a nonlinear activation function σ , which is needed to introduce non-linearities to the model. Commonly used activation functions are discussed in Section 4.1.3.

A perceptron is only capable of performing linear classification tasks. As the most real-world problems are very complex and require more than a linear classifier, more sophisticated models are used. One of them is the *multi-layer perceptron* which will be discussed in the following.

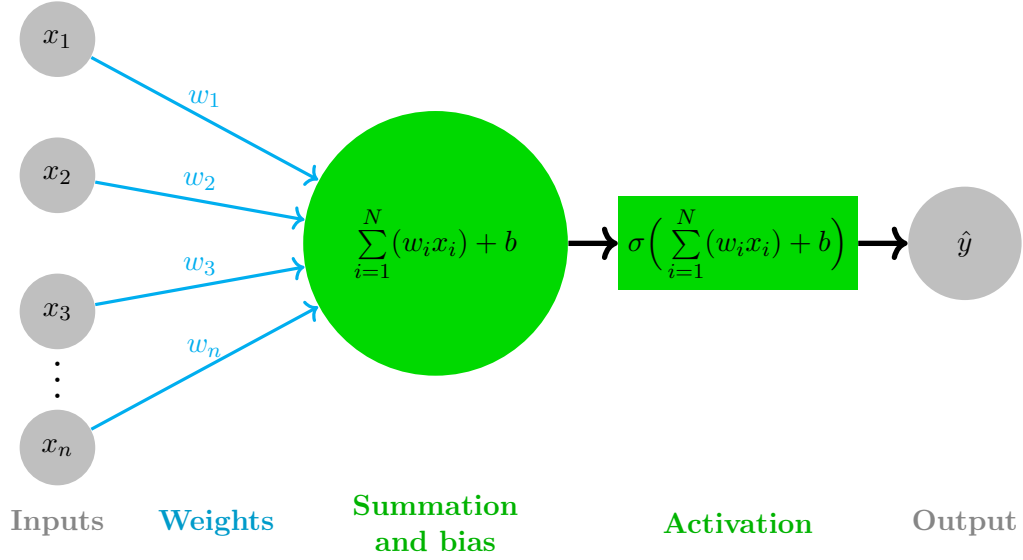


Figure 4.1: The simplest form of a neural network. Each input x_i is multiplied with a weight w_i . The sum over all products plus a bias term b is then passed to an activation function σ which yields the output $\hat{y}$ of the network. The components depicted in green encompass the scope of functionality of a single neuron.

4.1.2 Multi-layer Perceptron

One can now use the concept of a single perceptron to build a more advanced neural network. A *multi-layer perceptron* consists of an input layer and an output layer and at least one intermediate, so-called hidden layer. Each layer consists of several nodes as illustrated in Fig. 4.2. Except for the input nodes, each node in the network corresponds to a single neuron, which receives input from all nodes in the previous layer and outputs its activation to all neurons in the subsequent layer. Furthermore, every node-to-node connection comes with the weighting factor that is used for the weighted sum inside the neuron.

The output vector $\vec{a}^k = (a_1^k, \dots, a_m^k)^T$ of layer k , where m is the number of neurons in layer k , is given by

$$\vec{a}^k = \sigma(W^k \vec{a}^{k-1} + \vec{b}^k). \quad (4.1)$$

$W^k = (w_{ij}^k)$ is the $m \times n$ weight matrix with n being the number of neurons in layer $k-1$, $\vec{b}^k$ the bias vector and σ the activation function which is understood to be applied separately for every component of the vector.

Eq. (4.1) can also be written in a component-wise way in order to emphasize the behavior of a single neuron as introduced in the previous section. The activation output of neuron i in layer k is therefore given by

$$a_i^k = \sigma\left(\sum_{j=1}^n w_{ij}^k a_j^{k-1} + b_i^k\right). \quad (4.2)$$

The final output vector $\hat{\vec{y}}$ of the full network, i.e. the network's prediction for a given input vector $\vec{x}$, is obtained by using Eq. (4.1) successively for every layer (a so-called forward pass). Since the data flows in only one direction without forming cycles (as opposed to e.g. recurrent neural networks), these kind of networks are called *feed forward neural networks*. Neural networks with one or more hidden layers are also referred to as *deep neural networks*.

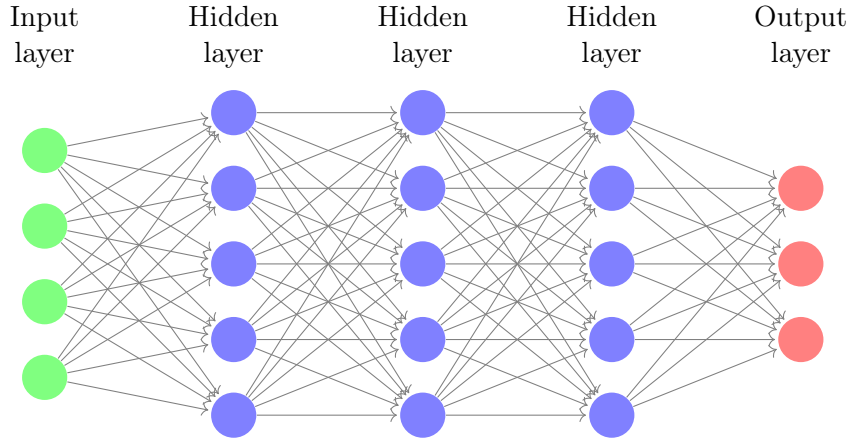


Figure 4.2: Multi-layer perceptron with four input nodes, three output nodes and with three hidden layers, each consisting of five neurons.

4.1.3 Activation Functions

Historically, a binary step function was used for the perceptron to decide whether to be activated or not. These theoretical considerations turned out not to be useful in practice especially due to the vanishing gradient for $x \neq 0$. As we will see in Section 4.1.4, a nonzero gradient of the activation function is needed for training the network. Therefore other activation functions have emerged and are extensively used in deep learning. The four most common activation functions are listed in Table 4.1. The choice of activation function mainly depends on the type of problem the network is supposed to solve.

If the network is supposed to solve a classification task, i.e. predicting a certain class the input belongs to, one usually uses a logistic function like the **sigmoid** activation function since its output can be treated as a probability. Additionally, its large gradient near zero shifts the outcome to the extremes, which is desired for class prediction. The **hyperbolic tangent** can be seen as a scaled version of the sigmoid function, centered around zero with outcomes between -1 and $+1$. Especially the mapping of negative values onto negative outputs and positive values onto positive outputs can be an advantage over the sigmoid function.

The most commonly used activation function in the context of Convolutional Neural Networks (see Section 4.2) is the **ReLU** activation function. It is also used for regression problems, i.e. predicting a certain value as opposed to a class probability, since

it has no upper bound. The vanishing gradient for $x \leq 0$ can sometimes be problematic, causing the so-called “dying ReLU problem”: Large weight updates can lead to a constant ReLU output of zero for any given input. It is unlikely to recover due to the gradient of zero for $x \leq 0$ and will remain “dead” for the rest of the training.

Leaky ReLU solves this problem by introducing a small slope for negative x . The parameter α is usually set to a small value in the order of $\alpha = 0.01$. In this way, the activation function can recover from the potential problem described above, making it a suitable alternative to ReLU.

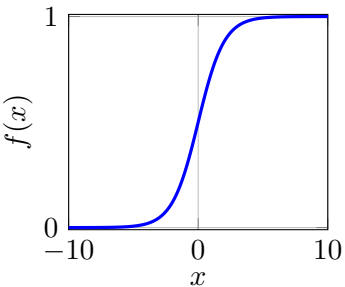
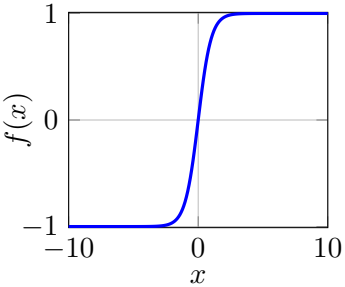
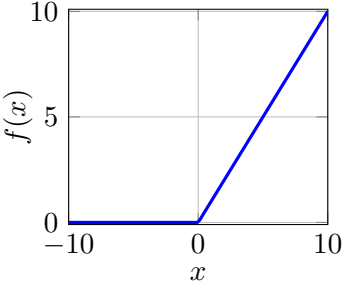
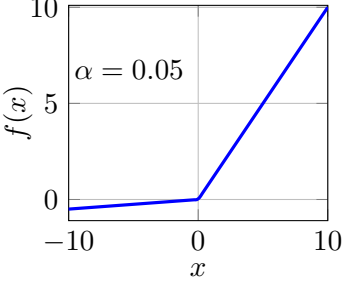
Name	Definition	Plot
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	
Tanh	$f(x) = \tanh(x)$	
ReLU	$f(x) = \max(0, x)$	
Leaky ReLU	$f(x) = \max(\alpha x, x), \alpha < 1$	

Table 4.1: Overview over the four most common activation functions in deep learning.

4.1.4 Gradient Descent and Backpropagation

Training a neural network, that is trying to find a set of weights that optimizes the network's performance, involves some sort of optimization algorithm [22]. Optimization usually refers to the process of minimizing a function $E: \mathbb{R}^n \rightarrow \mathbb{R}$ by altering its n variables. The function we want to minimize is a function that measures how well the network performs and will be called *loss function*. It is also known as *cost function* or *error function*.

Finding the absolute lowest value of the loss function, i.e. the global minimum, would of course be desirable, but is typically not possible in the context of deep learning. Especially if the function's parameter space is multidimensional, the occurrence of many flat regions and saddle points as well as poor local minima can make the optimization very difficult. Finding a set of weights that leaves the loss function in a local minimum that performs nearly as well as the global minimum is therefore an acceptable solution and the desired goal.

To find a local minimum of the loss function, one takes steps proportional to the negative gradient, as this is the direction where the value of the function decreases the fastest, until convergence. This method is called **gradient descent** and is schematically illustrated in Fig. 4.3.

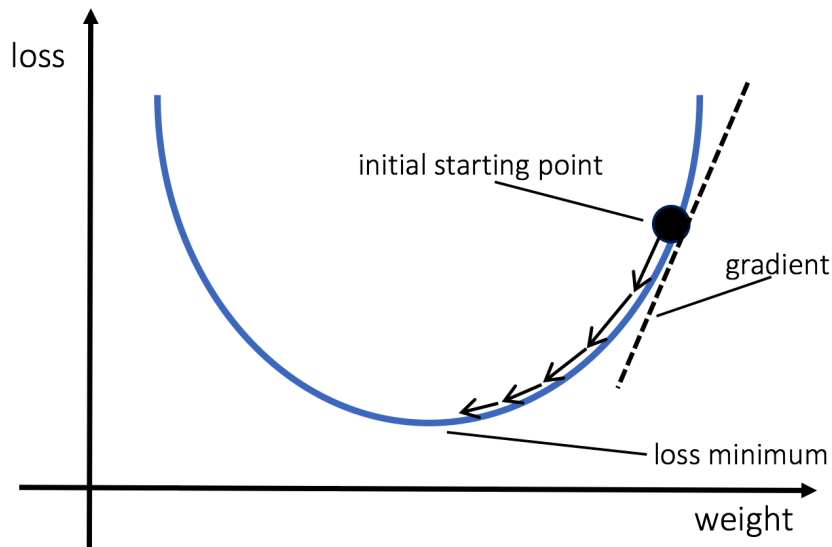


Figure 4.3: Schematic illustration of the gradient descent method for a loss function depending on only one single weight.

Using gradient descent to train a neural network requires the calculation of the gradient of the loss function $\nabla_{\theta} E(X, \theta)$ in order to update the neural network weights θ . A method called **backpropagation** is commonly used and will be explained in the following.

Three major components are necessary for the backpropagation algorithm:

1. A dataset $X = \{(\vec{x}_1, \vec{y}_1), \dots, (\vec{x}_N, \vec{y}_N)\}$ with N input-output pairs $(\vec{x}_i, \vec{y}_i)$, where $\vec{y}_i$ is the desired output of the network for input $\vec{x}_i$.

2. Weights w_{ij}^k and biases b_i^k , collectively denoted as θ , where w_{ij}^k is the weight between node j in layer k and node i in layer $k - 1$ and b_i^k the bias term for node i in layer k . Here, $k \in \{1, \dots, L\}$ where $k = 1$ represents the input layer and $k = L$ the output layer.
3. A loss function $E(X, \theta)$, which measures the difference between the desired output $\vec{y}_i$ and the current output of the network $\hat{\vec{y}}_i$. This output vector is calculated for the corresponding input $\vec{x}_i$ by performing a forward pass through the neural network with a set of current weights θ , as described in Section 4.1.1.

The weight updates at step t of the algorithm are then given by

$$\theta^t = \theta^{t-1} - \eta \nabla_{\theta} E(X, \theta^{t-1}). \quad (4.3)$$

The *learning rate* η determines how much the weights are updated in each iteration and thus plays an important role as one of the *hyperparameters* of the network (discussed in more detail in Section 4.4.4).

Consider the mean squared error as the loss function

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^N (\hat{y}_i - \vec{y}_i)^2, \quad (4.4)$$

where a factor of $1/2$ is added for convenience as seen below and let

- z_i^k be the summed products of weights and inputs plus bias of node i in layer k ,
- $a_i^k = \sigma_k(z_i^k)$ the output of neuron i with activation function $\sigma_k(z)$ in layer k and
- n_k the number of neurons in layer k .

In order to further simplify the following calculations, set

$$b_i^k \equiv w_{0i}^k. \quad (4.5)$$

The incorporation of the bias term b_i^k into the weights as w_{0i}^k with a fixed output $o_0^{k-1} = 1$ of node 0 in layer $k - 1$ can be seen as a redefinition of the raw activation z_i^k of neuron i in layer k . The calculation of the gradient requires the derivatives $\frac{\partial E}{\partial w_{ij}^k}$ which can be rewritten in the following way:

$$\frac{\partial E(X, \theta)}{\partial w_{ij}^k} = \frac{1}{N} \sum_{l=1}^N \frac{\partial}{\partial w_{ij}^k} \left(\frac{1}{2} (\hat{y}_l - \vec{y}_l)^2 \right) = \frac{1}{N} \sum_{l=1}^N \frac{\partial E_l}{\partial w_{ij}^k} \quad (4.6)$$

In the following derivation of $\frac{\partial E_l}{\partial w_{ij}^k}$, the index l will be omitted for simplicity. The final result for the gradient will be the average over the entire dataset X according to Eq. (4.6).

The derivation of the gradient with respect to the weight in the output layer $w_{ij}^k = w_{ij}^L$

is straightforward due to the direct influence on the error function.

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}^L} &= \frac{\partial}{\partial w_{ij}^L} \left(\frac{1}{2} \sum_{r=1}^{n_L} (\hat{y}_r - y_r)^2 \right) \\
&= -(\hat{y}_j - y_j) \frac{\partial}{\partial w_{ij}^L} y_j \\
&= -(\hat{y}_j - y_j) \sigma'_L(z_j^L) \frac{\partial}{\partial w_{ij}^L} z_j^L
\end{aligned} \tag{4.7}$$

Using now that

$$\begin{aligned}
\frac{\partial}{\partial w_{ij}^L} z_j^L &= \frac{\partial}{\partial w_{ij}^L} \sum_{i=0}^{n_{L-1}} \sigma_{L-1}(z_i^{L-1}) w_{ij}^L \\
&= \sigma_{L-1}(z_i^{L-1}) = a_i^{L-1}
\end{aligned} \tag{4.8}$$

leads to

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}^L} &= -(\hat{y}_j - y_j) \sigma'_L(z_j^L) a_i^{L-1} \\
&= \delta_j a_i^{L-1},
\end{aligned} \tag{4.9}$$

with the so-called error term defined as the output error of the network weighted by the derivative of the activation function in the output layer

$$\delta_j = -(\hat{y}_j - y_j) \sigma'_L(z_j^L). \tag{4.10}$$

According to Eq. (4.9), the amount of how much the loss function is affected by the weight w_{ij}^L is therefore given by the error term weighted with the magnitude of the activation from the previous node associated with the respective weight.

Similar calculations to the ones above can be done for the next-to-last layer ($k = L-1$) of the network. The gradient for the last hidden layer weights then takes a similar form to the one of the output layer with

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j a_i^{k-1}, \tag{4.11}$$

where

$$\delta_j = \sigma'_k(z_j^k) \sum_{r=0}^{n_L} \delta_r w_{jr}. \tag{4.12}$$

As one can see, the output error of the network is gradually being back-propagated through the network in order to determine the derivatives of the loss function with respect to the weights, hence the name of the algorithm.

By repeatedly applying the chain rule, these derivations can also be performed for all previous layers of the network, up to the input layer. In this way, the derivatives of the loss function with respect to every single weight in the network, and thus the gradient $\nabla_{\theta} E(X, \theta)$, can be found, allowing us to utilize gradient descent for training the neural network.

4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special class of deep feed-forward neural networks, designed for processing data with a grid-like topology. CNNs have been tremendously successful in fields like image classification or speech recognition by employing a mathematical operation called *convolution*, hence the name.

In place of the general matrix multiplication introduced in Section 4.1.2, a discrete convolution is used in at least one layer of a CNN instead. The underlying concepts are explained in the following [22].

4.2.1 Basic Concepts

Let x and w be complex-valued (real-valued in the context of CNNs) functions defined on the set $\mathbb{Z}$ of integers, the **discrete convolution** of x and w is then given by

$$s(t) = (x * w)(t) = \sum_{m=-\infty}^{\infty} x(m)w(t-m). \quad (4.13)$$

In deep learning terminology, $x(t)$ is usually referred to as the *input*, $w(t)$ as the *kernel* or *filter* and the output $s(t)$ is referred to as the *feature map* or *convolved feature*.

As the input is usually a multidimensional array of data, the kernel will also become a multidimensional array of parameters. These parameters correspond to the weight matrix entries in Section 4.1.2 and are adapted by the learning algorithm, i.e. the kernel is learned by the neural network.

For a two-dimensional input grid I , e.g. an image, one usually also uses a two-dimensional kernel K and defines the convolution of I and K according to

$$(I * K)(i, j) = \sum_m \sum_n I(i+m, j+n)K(m, n). \quad (4.14)$$

Note that the infinite sum of Eq. (4.13) is replaced by sums over finite sets of array elements. This is legitimate by assuming that the functions I and K are zero everywhere except for a finite set of points.

The following example will illustrate the concept of convolution as it is employed in a single convolution layer in a CNN. Fig. 4.4 shows an example input grid with size 5×5 and an example kernel with size 3×3 . The convolved feature is obtained by sliding the kernel across the entire input while multiplying corresponding entries and summing them up according to Eq. (4.14). This operation is illustrated in Fig. 4.5, resulting in a 3×3 feature map.

Often one wants to maintain the shape of the input for the feature map or obtain a feature map with a customized shape. Apart from the kernel size, there are two main parameters that can be modified in order to change the shape of the resulting feature map: **padding** and **stride**. Padding enlarges the input grid artificially as it

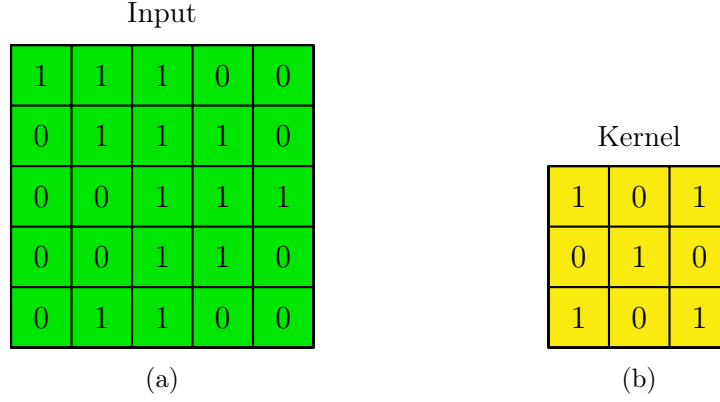


Figure 4.4: Examples of a 5×5 input grid (a) and a 3×3 kernel (b). Due to its structure, the kernel will especially be sensitive to x-shaped features in the input image.

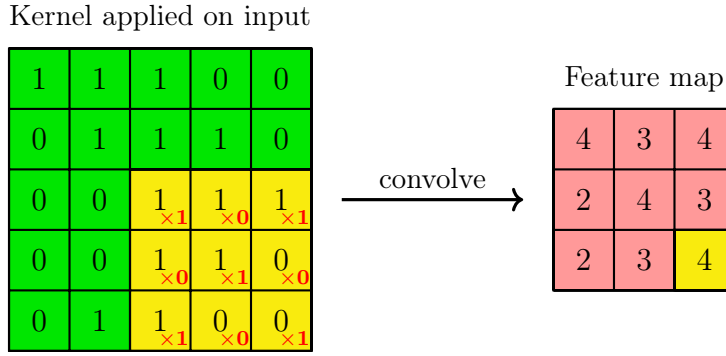


Figure 4.5: Convolving the input with the kernel from Fig. 4.4 results in a feature map. Here, the last step of the convolution process is shown, where the application of the kernel in the lower right corner of the input yields the lower right entry of the feature map.

pads the input with zeros around the border. Stride, on the other hand, controls how the kernel moves around the input during convolution. In the previous example, the stride S was implicitly set to $S = 1$ as the kernel shifts one step at a time.

Assuming a squared input and squared kernel for simplicity, the output size of the feature map can be calculated with

$$O = \frac{(W - K + 2P)}{S} + 1, \quad (4.15)$$

where O is the height and width of the resulting feature map, W the size of the input grid, K the kernel size and P the amount of zero padding.

So far, the input as well as the kernel were two-dimensional arrays. Such an input corresponds to a gray-scale image which does of course not reflect the most general case. Therefore a third dimension, the *channel* dimension, is introduced as a generalization. A usual colored image comprises three channels, a red, green and blue channel. Also the kernel becomes three-dimensional, where each channel is dedicated to detect features in the corresponding input channel. However, the depth of the

resulting feature map remains the same (if only one kernel is used) as the convolution outcomes of each channel are summed in depth.

Instead of using only one single (now three-dimensional) kernel in a convolutional layer, one can also utilize $k > 1$ kernels at a time. The application of k kernels then results in a feature map with k channels. Since the feature map is nothing but a new image, representing important features of the original image, it can be used as an input for the next convolutional layer.

This motivates the concatenation of many convolutional layers, which forms the foundation for the most common CNN architectures.

4.2.2 Convolutional Neural Network Architectures

The real strength of CNNs come with their depths as many concatenated convolutional layers of a deep network are able to extract various information and features and are thus able to analyze the input in its entirety. In order to reduce the amount of required computation in the network as well as the number of parameters (especially to prevent the network from overfitting, see Section 4.3.1), so-called pooling-layers are utilized.

Pooling is a form of down-sampling and an important concept in deep learning. The most widely-used pooling method is *max pooling*, whose functionality is illustrated in Fig. 4.6. The input grid is partitioned into several non-overlapping rectangles (depicted in different colors) and the maximum value of each sub-region is used as the output of the pooling operation. This procedure is justified by the idea that the relative positions of the most important features extracted by the pooling operation are more relevant than the exact locations of every single feature. This is especially true for classification problems. When dealing with e.g. object localization, pooling can of course be problematic since it worsens the resolution and positional information is lost.

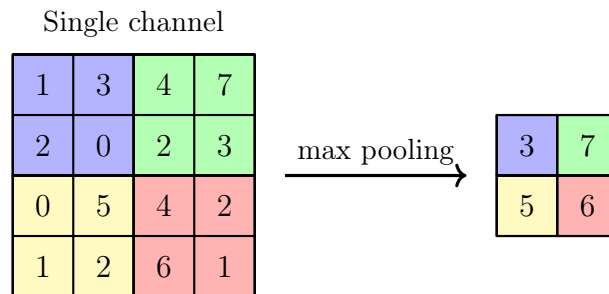


Figure 4.6: Example of a 2×2 max pooling operation with stride 2.

CNN Classification Architecture

Classification CNN architectures make extensive use of pooling layers. Fig. 4.7 shows the basis structure of such a CNN, which is almost always the same: A convolutional layer is followed by a pooling layer, which is in turn again followed by a convolutional

layer and so on. This pattern repeats several times, often until the spatial extent of the feature maps is so small that no pooling operation is applicable anymore. This type of down-sampling lets the network focus on *what* features are in the input, represented by the numerous feature maps, rather than *where* those features are. The last pooling layer is then followed by one or several dense layers (*fully-connected* layers) of neurons, meaning that every feature map entry is connected to every neuron in the subsequent layer. In the case of multi-class classification, the output layer consists of as many node as there are classes to distinguish between. Every output node then outputs the probability of the input belonging to the respective class.

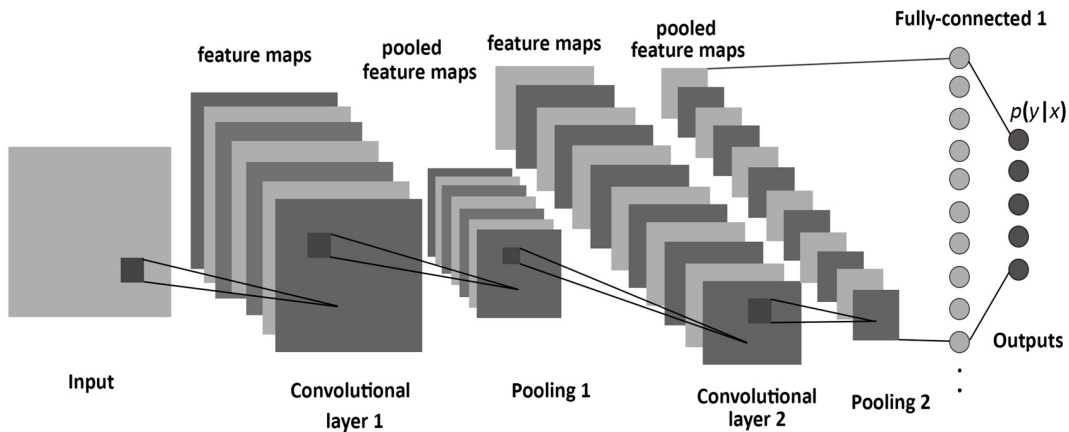


Figure 4.7: Common CNN architecture, consisting of convolutional, pooling and full-connected layers [23].

CNN Segmentation Architecture

Segmentation refers to the process of assigning each pixel in an image to a certain object class, the input image is thus segmented into several classes. Since the output of a segmentation CNN is also an image with the exact same size as the input, the general architecture will be different from the classification one. Fig. 4.8 shows a CNN used for segmentation, the so-called Deconvnet [24]. It basically consists of two separate networks, a convolutional network down-sampling the input into a feature vector and a deconvolutional network up-sampling the feature vector into a segmentation map. The feature vector is a compressed representation of the input image and hence contains all important information. The deconvolutional part of the network then uses these information to construct a class-segmented version of the input. Here, the two main operations employed for up-sampling are unpooling and deconvolution, also known as transpose convolution, as illustrated in Fig. 4.9.

Unpooling performs the reverse operation of pooling and reconstructs the original size. It uses so-called switch variables to record the locations of maximum activations during the pooling operation to place each activation back to its original pooled position in the unpooling step, while the remaining pixels in the unpooled map are set to zero. The sparse unpooling output is then densified using deconvolution. Deconvolution connects

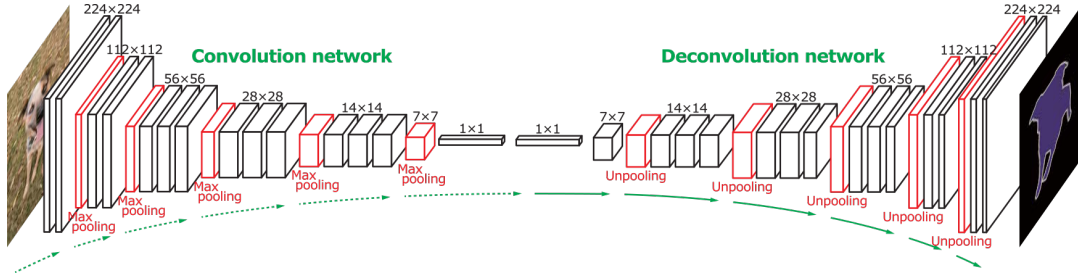


Figure 4.8: The Deconvnet architecture is composed of a down-sampling convolutional neural network and an up-sampling deconvolutional neural network, resulting in a segmentation map of the input image [24].

a single input activation to multiple outputs in contrast to ordinary convolution where several input activations within a kernel window are associated with one single output.

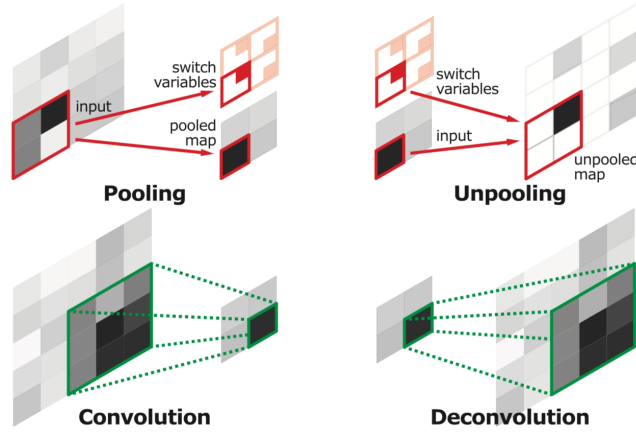


Figure 4.9: Illustration of deconvolution and unpooling operations [24].

Other up-sampling techniques involve alternating layers of strided deconvolution and ordinary convolution or other unpooling approaches, which do not use switch variables but fill the entire window in the unpooled map with the value of maximum activation in the unpooling step, also combined with ordinary convolutional layers.

A neural network like this, trained on reconstructing the input image itself, is called a *convolutional auto-encoder* as it first encodes all relevant information into a dense representation and then decodes these information to reconstruct the original image.

4.3 Regularization

In Section 4.1.4 the training process was described as finding a set of weights that minimizes the error function on the training dataset, the *training error*. A key challenge in machine learning is, however, that the model is able to *generalize*, i.e. it also performs well on new, previously unseen inputs. This is what separates a pure optimization problem from machine learning.

Therefore, the goal is to minimize the training error by simultaneously keeping the error on a new test dataset, the *test error* or *generalization error*, small. Many effective strategies for achieving this goal have emerged in the past, collectively known as *regularization*.

4.3.1 Underfitting and Overfitting

Two central aspects in machine learning are *underfitting* and *overfitting*, both of which causing poor performance of the algorithm. Underfitting refers to a model that is neither able to perform well on training data nor able to generalize to new data. However, underfitting can be fixed relatively simple by e.g. increasing the model's capacity. Overfitting, on the other hand, occurs when a model learns training-data specific details and noise that do not necessarily represent general properties, resulting in a low training error, but a high generalization error. Regularization methods aim to prevent the model from overfitting and the most important ones for this thesis will be discussed in the following.

4.3.2 Early Stopping

The maybe simplest form of regularizing a model is early stopping. This refers to the process of stopping the training when a gap between the test and training error starts to arise. Fig. 4.10 illustrates this method. After each training cycle, i.e. the network has seen the entire training dataset once, test and training errors are evaluated. At the point right before the model tends to overfit (dashed line), the training is stopped in order to obtain a well generalizing model.

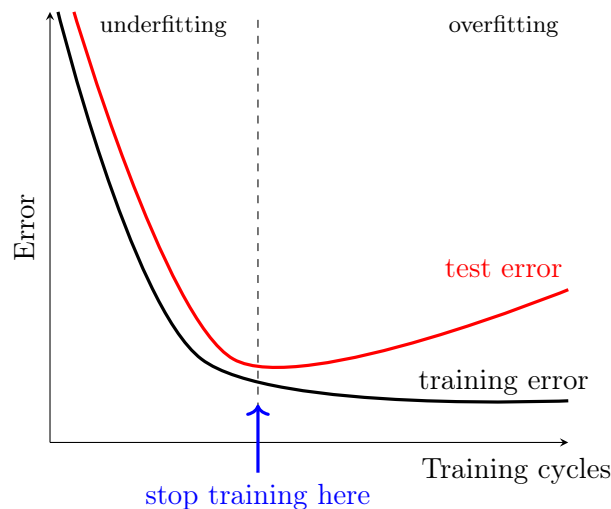


Figure 4.10: Illustration of a typical error development over training cycles. Test and training errors first decrease, at a certain point the test error starts increasing into overfitting. The dashed line separates the underfitting region (left) from the overfitting region (right).

4.3.3 Dropout

Dropout is a really simple, yet very effective regularization technique. It refers to ignoring (dropping out) a random set of neurons during training time. One defines a dropout probability factor p that determines how likely it is for a single neuron to be dropped out at each training iteration. A dropped-out neuron is then ignored during the training forward and backward pass. This prevents units and their weights from adapting too much to the training data and therefore helps the model to generalize better.

Since the network lacks a certain amount of its neurons during training but uses the full network architecture for evaluation, the loss for the training dataset will generally be larger than the one for test dataset.

4.3.4 Parameter Norm Penalties

Other regularization techniques are also based on the idea that a few specific weights should not have too much influence on the model's outcome and thus impose certain restrictions on trainable parameters in the network. This can be accomplished by adding a parameter norm penalty term $\Omega(\theta)$ to the loss function E . The regularized loss function is denoted by $\tilde{E}$:

$$\tilde{E}(X, \theta) = E(X, \theta) + \lambda \Omega(\theta), \quad (4.16)$$

where λ is a hyperparameter that weights the relative contribution of the norm penalty to the overall loss function.

L^2 -Parameter Regularization

One of the simplest and most common parameter norm penalty terms is the L^2 norm. This regularization method pushes the weights closer to small values by adding the penalty term $\Omega(\theta) = \frac{1}{2} \|\vec{w}\|_2^2$ to the loss function, where $\vec{w}$ denotes a vector containing all weights of the model. The total error function thus becomes:

$$\tilde{E}(X, \theta) = E(X, \theta) + \frac{\lambda}{2} \vec{w}^T \vec{w}. \quad (4.17)$$

One important feature of L^2 regularization is its computational efficiency due to having analytic solutions for the gradient.

L^1 -Parameter Regularization

Another option is to use L^1 regularization instead, with

$$\Omega(\theta) = \|\vec{w}\|_1 = \sum_i |w_i|. \quad (4.18)$$

This leads to the total loss function

$$\tilde{E}(X, \theta) = E(X, \theta) + \lambda \sum_i |w_i|. \quad (4.19)$$

Despite having no analytic solution for the gradient, L^1 regularization features some useful properties. It results in a more *sparse* outcome in comparison to L^2 , meaning that less important parameters are pushed to have a value of exactly zero. Only parameters representing very important features remain nonzero, which can be extremely useful if small and compact models are required.

4.4 Training Optimization

Optimizing the training of a neural network in terms of finding the best possible local minimum but also in terms of computational efficiency is one of the key challenges in deep learning as discussed below.

4.4.1 Batch and Minibatch Algorithms

Training a neural network by calculating the gradient over the entire dataset as described in Eq. (4.6) is called **batch training**. This method is very memory-consuming and thus often not possible. Additionally, the large average over a vast majority of potentially very important information often leads to saddle point solutions and is thus not optimal.

As opposed to building the gradient over the entire dataset, the other extreme approach is **stochastic gradient descent**, where the gradient is computed on single instances which are randomly picked from the dataset. Since this method is based on random data points, it is very sensitive to noise and only works with very small learning rates.

However, noise can be a great advantage as it helps to escape from poor local minima or saddle points. Therefore, most deep learning optimization algorithms use more than one but less than all training examples for calculating the gradient at one iteration, which is known as **minibatch training**. Especially when training on GPUs, better runtimes are achieved by using power of two batch sizes¹, often ranging from

¹Note that “batch size” always refers to the size of a minibatch in minibatch training and is not to be confused with the word “batch” in batch training.

32 to 256.

Small batches can also have a regularization effect (see [25]) and are therefore the most common choice.

4.4.2 Parameter Initialization

The initial parameter starting point for training a neural network determines whether the algorithm converges at all, how quickly it converges and also if it converges to a local minimum with high or low loss. A proper initialization of the trainable parameters is therefore crucial for an effective training.

For a weight matrix with n parameters, a common choice is to sample initial weights from a normal distribution centered around zero with a standard deviation of $\sigma = \sqrt{\frac{2}{n}}$, truncated at 2σ .

4.4.3 Input Data Preparation

Data pre-processing is necessary since large variances in the input data lead to different scales of different features in the loss function. As a result, the gradient is too small in some regions to converge in a reasonable time and too big in other regions to converge at all.

The two mainly used methods are:

1. **Min-max normalization** where the input data is scaled into an interval of $[0, 1]$ or $[-1, 1]$. This transformation retains the original distribution shape but is heavily sensitive to outliers.
2. **Standardization** which scales the inputs to have a mean of $\mu = 0$ and a variance of $\sigma^2 = 1$, but does not guarantee a common numerical range of the standardized data.

When dealing with regression problems where features in the output have different scales, a normalization of the output features is necessary to ensure that each feature contributes with the same scale to the gradient. Otherwise the training would suffer from a gradient that is very large in some regions and very small in others.

4.4.4 Hyperparameter Optimization

Hyperparameters are parameters whose values are set by the user before the training process begins and are not adapted by the learning algorithm itself. Hence, they allow to control the algorithm's behavior and need to be optimized.

Hyperparameters come in two categories:

- Model-specific hyperparameters, such as the amount of hidden layers in the network, the amount of units per layer, the kernel size or the amount of kernels in a CNN.
- Training-specific hyperparameters, such as the learning rate η , the batch size or the regularization parameter λ .

In order to find the best possible set of hyperparameters, a **validation dataset** is needed, that is not observed by the training algorithm.

In Section 4.3 the *test dataset* was introduced to estimate the generalization error of the algorithm. Since this dataset is used for evaluating the algorithm's performance on unseen data after training, it is important that examples from the test dataset are not used to make any choices about the model, including hyperparameters. Therefore a third dataset, the validation set, is constructed by splitting the training set into two disjoint subsets. Typically, one uses about 80% of the training set for the actual training, while the other 20% are used to optimize the hyperparameters.

Finding an optimal set of hyperparameter then often involves grid searches for evaluating different hyperparameter sets on the validation data. After optimizing the hyperparameters, the trained model can be evaluated on the test dataset.

Chapter 5

Monte Carlo Simulation Samples

Training a deep neural network requires a huge dataset to learn from with millions of examples. This data has to be chosen wisely, representing all important characteristics for the task the network has to solve.

The work presented in this thesis is based on two different types of datasets:

- **Fast simulation data**, used for the training of the neural networks
- **Full ATLAS Monte Carlo data**, used for evaluation of the networks

The generation as well as relevant properties of these datasets will be covered in this chapter.

5.1 Fast Simulation Data

Simulating a full high-energy physics event normally consists of two stages, event generation and detector simulation. The event generator describes the particle reactions and produces momentum four-vectors of the final-state particles that make up the event. This output is then passed to a detector simulation program, which usually uses the entire detector description with its geometrical elements, materials and electronics properties to simulate the detector response to the given input.

Especially the detector simulation is computationally very expensive such that the full simulation of one single event can take up to several minutes.

Since the required amount of training data is not available as full ATLAS Monte Carlo data (see Section 5.2), a fast simulation is employed instead. Fast simulation refers to simulating events using an ordinary event generator for producing final-state particles as described above but instead of utilizing a detector simulation program, a very simplistic form of detector response simulation is used.

Besides being able to produce as much data as needed in a reasonable period of time, a customized fast simulation also enables the possibility of controlling and adjusting important physics properties of the training data. One of these properties is the pile-up $\langle\mu\rangle$, i.e. the amount of additional proton-proton interactions in a bunch crossing accompanying the interesting hard scattering process. These additional interactions

are added as minimum-bias events, which are basically all occurring processes with cross sections usually orders of magnitude larger than the actual hard scattering process cross section.

The fast simulation data is thus produced in two successive steps: Event generation with the **PYTHIA** [5] event generator, followed by a fast detector response simulation. These two steps are explained in the following.

5.1.1 Event Generation with PYTHIA

Like every typical hadronic event generator **PYTHIA** uses simple perturbative tree-level quantum field theory to describe collisions and decay processes in a high energy physics event. In order to account for the significant amount of usually occurring radiative loop corrections and other modifications like photon or gluon bremsstrahlung, the full process is factorized into individual sub-processes which can be calculated separately. The probabilistic branching between these processes is determined using Monte Carlo methods.

The following sub-processes are simulated:

- Composition and substructure in the initial state
- Initial-state radiation
- Hard scattering process
- Resonance decays
- Final-state radiation
- Additional semi-hard processes
- Hadronization and further decays

Since the neural network will eventually be trained on finding jets using this training dataset, physics processes that yield a large amount of hadronic jets are chosen. The physics process of choice for the training data is therefore $t\bar{t}$ production.

Fig. 5.1 shows the Feynman diagrams for the two dominant $t\bar{t}$ production modes at the LHC in the $gg \rightarrow t\bar{t}$ channel.



Figure 5.1: The most dominant channels for $t\bar{t}$ production at the LHC.

Due to their high masses, top quarks decay almost instantaneously before being able to hadronize. The main decay channel of the top quark is via the electroweak interaction into a W boson and a bottom quark. The two resulting bottom quarks of the $t\bar{t}$ decay lead to two jets and also the W bosons are likely to produce jets (branching ratio of about 67.6% [26]) which are all measured in the calorimeters.

In addition, a variable amount of minimum-bias processes, accounting for pile-up, can be added on top of the $t\bar{t}$ production.

5.1.2 Fast Detector Response Simulation

Every final-state particle of the event generation described above that interacts with the calorimeter material (i.e. hadrons, photons, electrons and positrons) produces a particle shower as it traverses through the detector. The simplistic detector response simulation is based on the assumption that any of those particles of the **PYTHIA** event will produce a Gaussian-shaped particle shower in the detector. Since muons lose only a small fraction of their energy in the calorimeter, they are not taken into account here.

The lateral profile of a particle shower is highly depended on the calorimeter characteristics. For electromagnetic showers the lateral extent is governed by the Molière radius R_M . By definition, it is the radius of a cylinder containing an average of 90% of the shower's energy deposition in the calorimeter. The Molière radius of the ATLAS LAr calorimeter is given by $R_{M,em} = 4.8$ cm [27], whereas the hadronic shower radius for 95% radial containment for the ATLAS Tile calorimeter is $R_{had} = 25.1$ cm [28]. As one can see, hadronic showers are typically much broader than electromagnetic ones. The lateral extent of particle showers in terms of η and ϕ can be approximated as explained in the following.

Lateral Electromagnetic Shower Profile Approximation

Using the inner and outer radius of the LAr calorimeter as describes in Section 3.3.2, the average depth of the electromagnetic calorimeter at $\eta = 0$ is given by

$$\bar{r}_{em} = 115 \text{ cm} + \frac{225 - 115}{2} \text{ cm} = 170 \text{ cm}. \quad (5.1)$$

Therefore, an azimuth region of $\Delta\phi = 0.1$ corresponds to a calorimeter cell height of approximately

$$d_\phi = \frac{2\pi\bar{r}_{em}}{64} \approx 16.7 \text{ cm}, \quad (5.2)$$

where 64 is the number of calorimeter cells giving the full 2π azimuth coverage.

Analogously, the width of a calorimeter cell at $\eta = 0$ corresponding to $\Delta\eta = 0.1$ is given by

$$d_\eta = \bar{r}_{em} \tan\left(\frac{\pi}{2} - \theta\right) = \bar{r}_{em} \tan\left(\frac{\pi}{2} - 2 \arctan(e^{-\Delta\eta})\right) \approx 17.1 \text{ cm}. \quad (5.3)$$

For simplicity, squared calorimeter cells of $d_\eta \times d_\phi \approx 17 \times 17 \text{ cm}^2$ are assumed. 90% of the shower's energy is thus contained in a circle with a radius of

$$\Delta\eta_{\text{em}} = \Delta\phi_{\text{em}} = \frac{0.1}{17 \text{ cm}} \cdot 2R_{\text{M,em}} \approx 0.06. \quad (5.4)$$

The calorimeter response information for a single incident final-state particle are obtained by a two-dimensional integration over a Gaussian distribution while filling the grid of calorimeter towers with the respective energy depositions. The width of the distribution is determined by the lateral shower size in such a way that 1.64σ correspond to the size given by Eq. 5.4, containing 90% of the shower's total energy. The standard deviation of the Gaussian distribution in terms of η and ϕ can therefore be calculated as

$$\sigma_\eta = \sigma_\phi = \frac{\Delta\eta_{\text{em}}}{1.64} = \frac{\Delta\phi_{\text{em}}}{1.64} = 0.03. \quad (5.5)$$

Since a two-dimensional integration for every final-state particle is computational very expensive and inefficient, pre-calculated shower profiles are used instead for simulation. For this purpose, a 11×11 binning of the calorimeter towers is introduced and the associated 121 shower profiles are calculated.

Fig. 5.2 shows two shower profiles for two different positions¹ of the incident particles. A particle that is centered with respect to a calorimeter tower produces a symmetric energy² deposition distribution (Fig. 5.2(a)), whereas an incident particle that falls into a bin off the center will yield an asymmetric distribution (Fig. 5.2(b)).

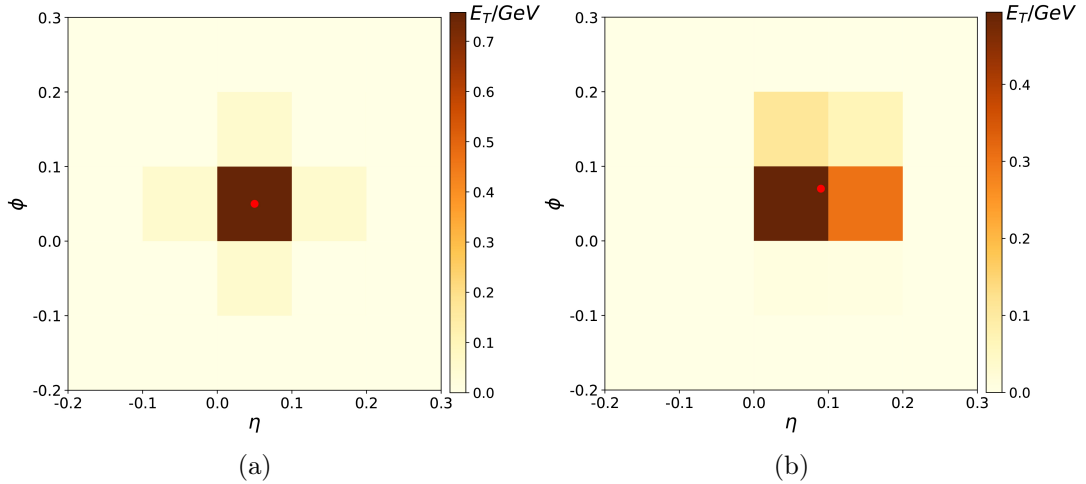


Figure 5.2: Electromagnetic shower profiles in the calorimeter for two different positions (red dots) of a 1 GeV incident final-state particle.

These shower profiles are used for all final-state electrons, positrons and photons of the event simulation.

¹“Position” refers to the angular position of the particle or jet in the calorimeter.

²When speaking about energies in the calorimeter, the terms “energy” and “transverse energy” are used interchangeably, but always mean the transverse energy E_T .

Lateral Hadronic Shower Profile Approximation

The lateral hadronic shower profile can be approximated analogously to the derivation of the electromagnetic shower profile above.

At $\eta = 0$, the average depth of the hadronic calorimeter is given by

$$\bar{r}_{\text{had}} = 228 \text{ cm} + \frac{423 - 228}{2} \text{ cm} = 325.5 \text{ cm}. \quad (5.6)$$

An azimuth region of $\Delta\phi = 0.1$ thus corresponds to a height of

$$d_\phi = \frac{2\pi\bar{r}_{\text{had}}}{64} \approx 32.0 \text{ cm} \quad (5.7)$$

and $\Delta\eta = 0.1$ to a width of

$$d_\eta = \bar{r}_{\text{had}} \tan\left(\frac{\pi}{2} - \theta\right) = \bar{r}_{\text{had}} \tan\left(\frac{\pi}{2} - 2 \arctan(e^{-\Delta\eta})\right) \approx 32.6 \text{ cm}. \quad (5.8)$$

Again, squared calorimeter cells of size $32.3 \times 32.3 \text{ cm}^2$ are assumed for simplicity in the following calculations.

The lateral extent of hadronic showers in terms of η and ϕ is therefore given by

$$\Delta\eta_{\text{had}} = \Delta\phi_{\text{had}} = \frac{0.1}{32.3 \text{ cm}} \cdot 2R_{\text{had}} \approx 0.15. \quad (5.9)$$

The standard deviation of the Gaussian distribution is also obtained in a similar way, where now 95% of the shower's energy correspond to the 2σ interval of the distribution.

$$\sigma_\eta = \sigma_\phi = \frac{\Delta\eta_{\text{had}}}{2} = \frac{\Delta\phi_{\text{had}}}{2} = 0.075. \quad (5.10)$$

Fig. 5.3 shows two hadronic shower profiles for two incoming hadrons with different relative positions to the calorimeter tower center. Analogue to the electromagnetic shower approximation, these shapes are used for all final-state hadrons in the event.

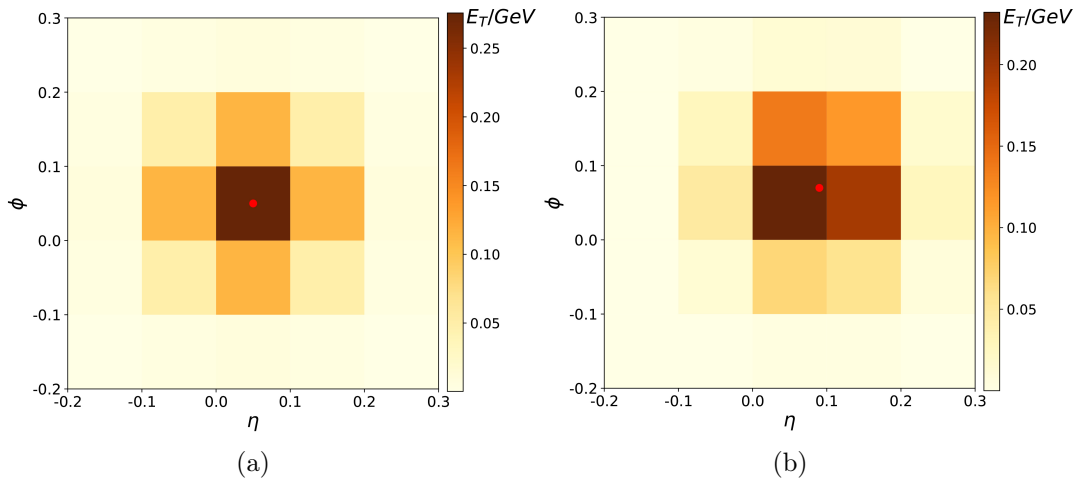


Figure 5.3: Hadronic shower profiles in the calorimeter for two different positions (red dots) of a 1 GeV incident final-state particle.

Final Summation and Data Format

Each event typically consists of several hundreds of particles in the final-state. The particle's positions and momenta are determined by the event generator. In order to obtain the response simulation of the entire calorimeter, the shower profiles described above are weighted with the particle's energy and summed up at the respective angular positions in the calorimeter.

After all energy depositions are summed up, the calorimeter energy resolution is taken into account: Each calorimeter tower entry $E_{T,ij}$ is newly determined by a Gaussian-distributed random number, centered around the respective energy value and with a standard deviation σ corresponding to the trigger tower's energy resolution. The trigger tower resolution is worse than the average calorimeter energy resolution (see Table 3.1) and is assumed to be 50%.

$$E_{T,ij} \rightarrow \text{RandomNormal}(\mu = E_{T,ij}, \sigma = 50\% \sqrt{E_{T,ij}}) \quad (5.11)$$

Fig. 5.4 shows an example fast simulation event display of the calorimeter in the region $|\eta| < 3.2$. The event is a pure $t\bar{t}$ event without any pile-up.

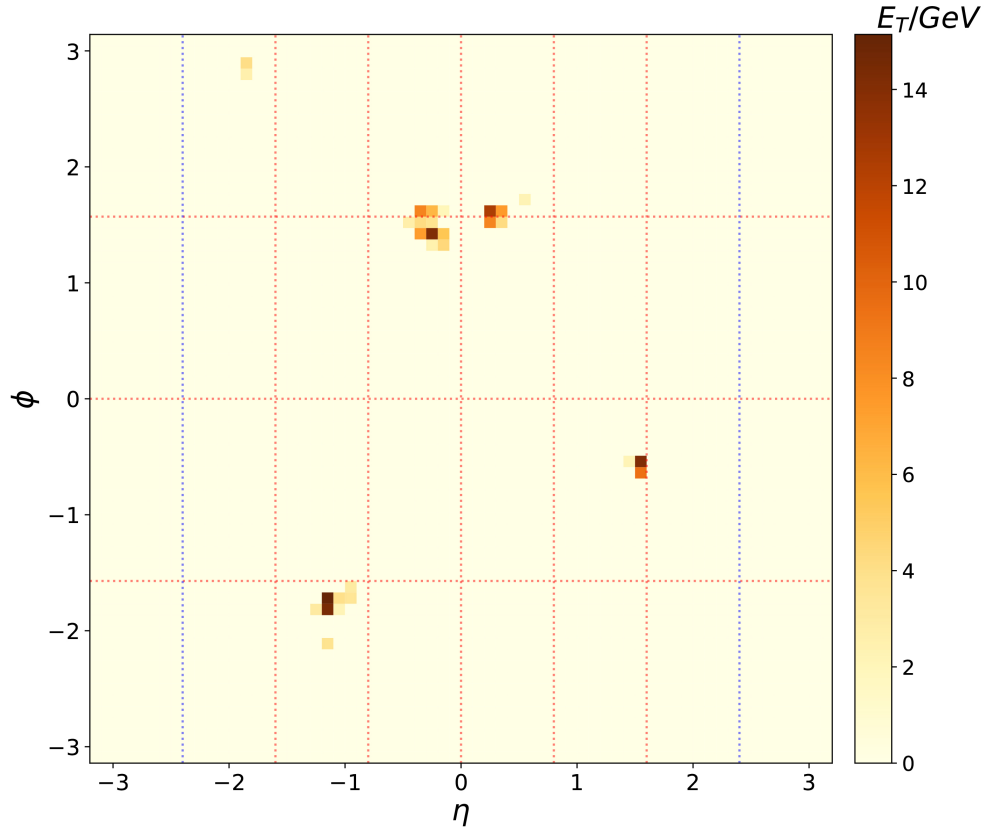


Figure 5.4: Fast simulation event display of a pure $t\bar{t}$ event without pile-up. The red dotted lines indicate the subdivision of the calorimeter into the jFEX FPGA subregions as described in Section 3.4.2. The two blue dotted lines at $|\eta| = 2.4$ mark the boundaries to the forward region with a more coarse granularity.

The data input to the neural network will have the format of an FPGA subregion as already shown in Fig. 3.8(a), i.e. 24×32 towers in $\eta \times \phi$. One single event like the one in Fig. 5.4 is therefore used to produce multiple smaller events, each of which representing a single FPGA subregion including the overlap with neighboring subregions. The outer regions with $|\eta| > 2.4$ will be omitted due to their coarse granularity. In addition, the neighboring FPGA subregions with $1.6 < |\eta| < 2.4$ have overlaps with these coarse granularity areas and will therefore not be used for training either. Thus, only the innermost 4×4 regions are used, giving 16 FPGA training events per $t\bar{t}$ event. In the process of subdividing the calorimeter into training data events, the wrap-around in ϕ is taken into account as a periodic boundary. Each new event has its own local coordinate system such that the lower left corner of the FPGA subregion is located at $(\eta, \phi) = (0, 0)$.

Fig. 5.5 shows two example events produced from the $t\bar{t}$ event of Fig. 5.4. The azimuthal wrap-around is apparent in Fig. 5.5(b).

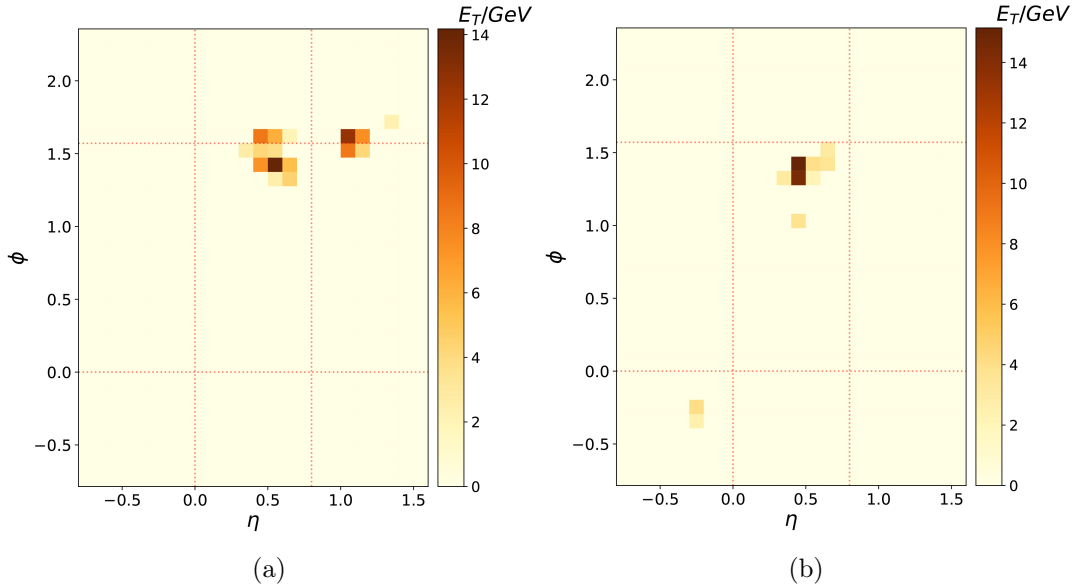


Figure 5.5: Example training data event displays with FPGA subregion format produced from the $t\bar{t}$ event of Fig. 5.4. Event (a) is obtained from the subregion with the global coordinates of the lower level corner of $(\eta, \phi) = (-0.8, 0)$ and event (b) from the subregion at $(\eta, \phi) = (-1.6, -\pi)$.

5.1.3 Anti- k_t Online and Anti- k_t Truth

A dataset $X = \{(x_1, y_1), \dots, (x_N, y_N)\}$ always consists of input-output pairs (x_i, y_i) as discussed in Section 4.1.4. So far, only the inputs x_i to the neural network, i.e. the fast simulated 24×32 grids of calorimeter towers, were covered.

The corresponding outputs y_i are obtained by utilizing the anti- k_t algorithm, meaning that the neural network will be trained on anti- k_t data. There are two different possibilities of employing the algorithm, differing in the choice of input objects:

1. The first option is to use calorimeter towers as input objects. The jet clustering will then be performed based on these discretized calorimeter information. This would be the method of choice for online reconstruction in the calorimeter trigger if available resources and latency allowed a utilization of this highly iterative algorithm (which is not the case). Hence, this approach is referred to as **anti- k_t online** and can be used as a benchmark for jet reconstruction in the Level-1 calorimeter trigger.
2. Instead of using discretized coarse calorimeter information, Monte Carlo simulation also allow to make use of the full available particle level information. The input objects used for clustering are thus momentum four-vectors of all final-state particles, leading to the best possible description and resolution of the physics event. This method will be called **anti- k_t truth**.

The anti- k_t truth method with a radius parameter of $R = 0.4$ and a jet p_t threshold of $p_t^{\text{cut}} = 20 \text{ GeV}$ will be used for generating the desired outputs y_i for the neural network training data. Unless stated otherwise, “anti- k_t ” will always refer to anti- k_t truth in the following.

Fig. 5.6 shows the event displays from Fig. 5.5 together with the jet energies and positions determined by anti- k_t .

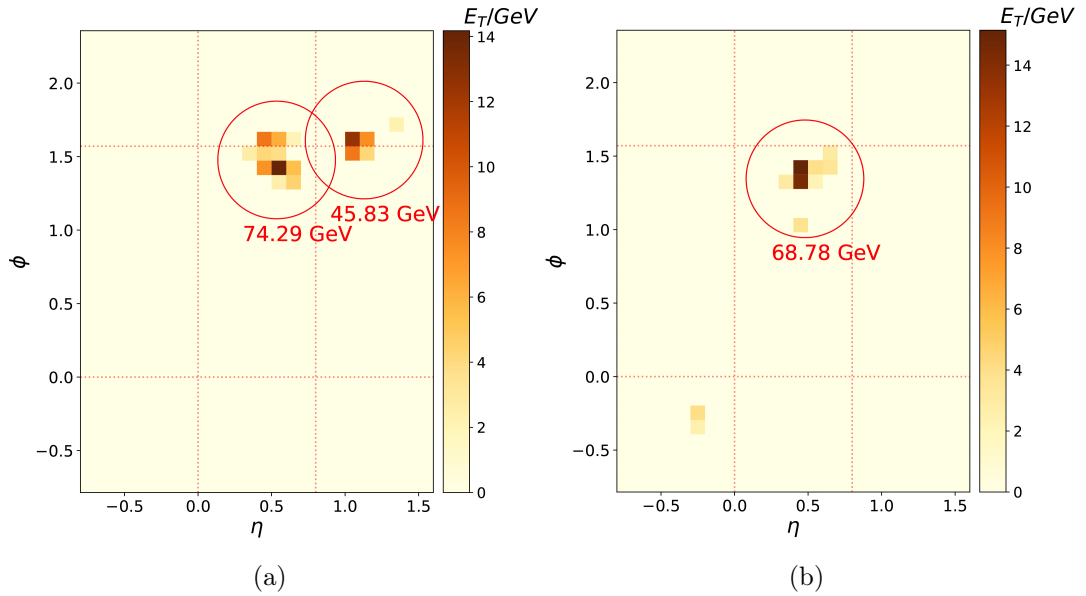


Figure 5.6: Two example event displays from Fig. 5.5 together with jets found by the anti- k_t algorithm, represented by red circles with a radius of $R = 0.4$, centered at the anti- k_t jet position.

More precisely, anti- k_t yields information about η and ϕ of the jet center as well as its total transverse energy E_T for every jet in the event.

For example, the anti- k_t jet information resulting from the two events of Fig. 5.6 are given in Table 5.1. The neural network outputs y_i will somehow represent these anti- k_t jet information, however, the exact format of y_i is dependent on the network’s architecture and will therefore be discussed in Section 6.1 and 6.2 respectively.

Event	Anti- k_t Jet Information
(a)	$(\eta_1, \phi_1, E_{T,1}) = (0.53, 1.48, 74.29 \text{ GeV})$
	$(\eta_2, \phi_2, E_{T,2}) = (1.13, 1.61, 45.82 \text{ GeV})$
(b)	$(\eta_1, \phi_1, E_{T,1}) = (0.48, 1.35, 68.78 \text{ GeV})$

Table 5.1: Anti- k_t jet information for the two events in Fig. 5.6(a) and Fig. 5.6(b).

5.2 Full ATLAS Monte Carlo Data

While the training requires millions of events and is thus entirely based on fast simulation data, the evaluation of the network’s performance can be done with only a few thousand events.

Full ATLAS Monte Carlo data will be used for this purpose since this data is simulated exploiting the entire detector description. The detector simulation is based on Geant4 [29] and incorporates for example

- detector geometry,
- detector materials,
- tracking through matter and magnetic fields and
- responses of sensitive detector components.

5.2.1 $hh \rightarrow 4b$

In order to guarantee that the neural network has learned to universally reconstruct jets, rather than only the physics of a $t\bar{t}$ event, another physics process will be used for evaluating the network’s reconstruction performance.

The $hh \rightarrow 4b$ channel features four jets in the final state which makes it a good process for producing evaluation datasets. Fig. 5.7 shows the two most dominant production modes for Di-Higgs production at the LHC.

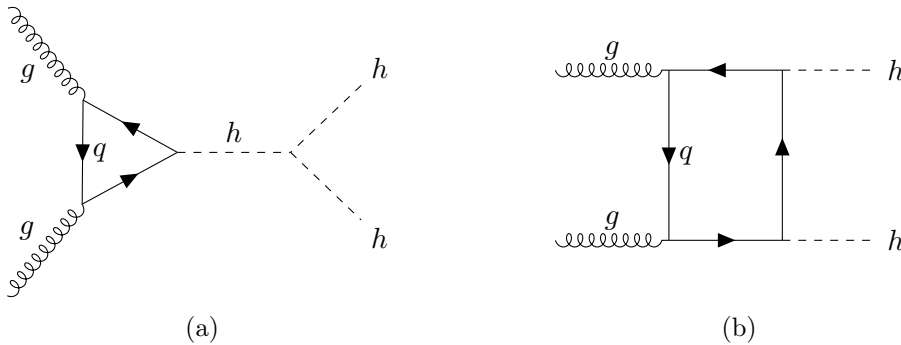


Figure 5.7: The two most dominant Di-Higgs production channels at the LHC.

Each Higgs boson then decays to a $b\bar{b}$ pair, giving four b-jets in the final state. A high p_T Higgs particle leads to a boosted pair of b-jets which will only have a small angular separation in the calorimeter. This is desired as it makes the jets overlapping and the reconstruction more challenging.

Fig. 5.8 shows an ATLAS full simulation Di-Higgs event display at 14 TeV center-of-mass energy and pile-up $\langle\mu\rangle = 60$.

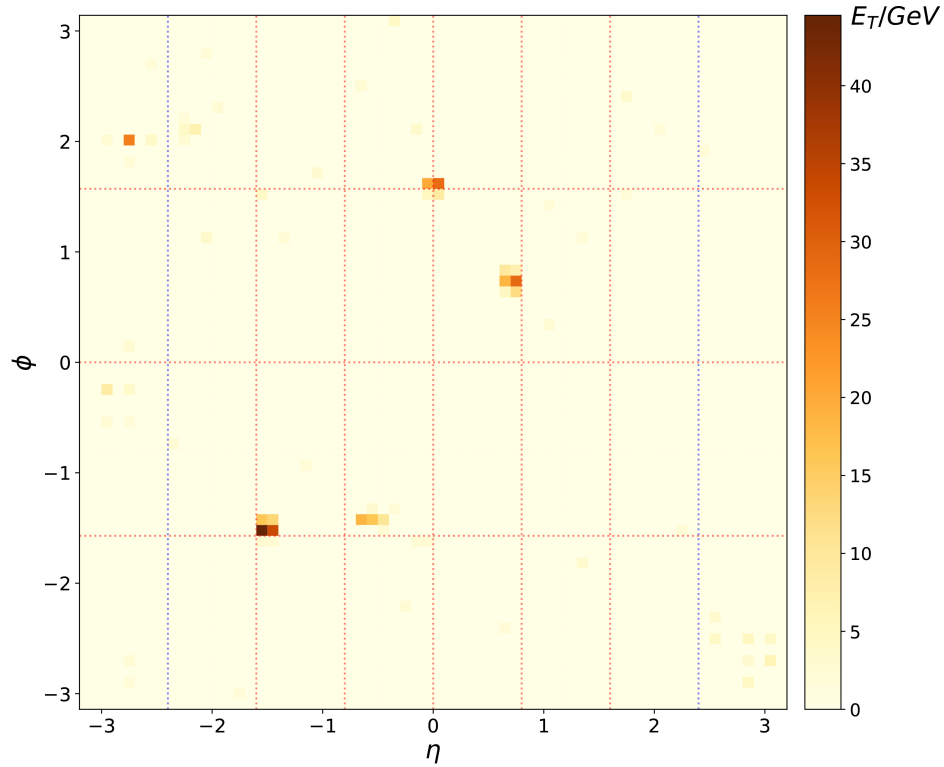


Figure 5.8: ATLAS full simulation event display of a $hh \rightarrow 4b$ event at 14 TeV center-of-mass energy and $\langle\mu\rangle = 60$.

Besides anti- k_t truth and anti- k_t online, offline reconstructed jets, i.e. reconstructed exploiting the full detector information, are available for ATLAS full simulation data, which will be referred to as **anti- k_t offline** jets.

5.2.2 $gg \rightarrow t\bar{t}$

Even though mainly Di-Higgs data will be used for evaluation of the neural network, a small full simulation $gg \rightarrow t\bar{t}$ dataset can be helpful for various tasks. This includes comparisons of the network's performance between fast and full simulation data with the same physics process (and thus allows to assess the training data) or comparisons between both full simulation samples to see if the network operates completely independent of the underlying physics process.

This ATLAS full simulation $t\bar{t}$ dataset is produced at 13 TeV center-of-mass energy and with $\langle\mu\rangle = 80$.

5.3 Overview over Available Data and Data Selection

Table 5.2 gives an overview over all available datasets together with their main properties.

As described in Section 4.4.4, the entire training dataset is divided into two disjoint subsets. About 80% of the training data is used for the actual training (“training” dataset), while 20% are held back for hyperparameter optimization (“validation” dataset). Additionally, a small $t\bar{t}$ fast simulation test dataset (“evaluation”) which will not be observed by the network during training is generated. It can be used for comparisons of the network performance between fast simulation and ATLAS full simulation test data.

There are two equally sized training sets, one without and one with pile-up ($\langle\mu\rangle = 80$). The most challenging part of jet reconstruction arises if two jets are close to each other and overlap. Jets are classified as *close-by* if the jet centers determined by anti- k_t are closer than $\Delta R < 0.6$. In order to specifically optimize the network’s performance on close-by jet reconstruction, datasets with FPGA subregion events containing close-by jets are constructed from training, validation and evaluation datasets.

Function	Physics process	Full Sim.	Selection	$\langle\mu\rangle$	CM energy	Quantity
training	$gg \rightarrow t\bar{t}$	no	all	0	14 TeV	10^8
		no	close-by	0	14 TeV	$4 \cdot 10^6$
		no	all	80	14 TeV	10^8
		no	close-by	80	14 TeV	$4 \cdot 10^6$
validation	$gg \rightarrow t\bar{t}$	no	all	0	14 TeV	$2.5 \cdot 10^7$
		no	close-by	0	14 TeV	10^6
		no	all	80	14 TeV	$2.5 \cdot 10^7$
		no	close-by	80	14 TeV	10^6
evaluation	$gg \rightarrow t\bar{t}$	no	all	0	14 TeV	$2.5 \cdot 10^6$
		no	close-by	0	14 TeV	10^5
		no	all	80	14 TeV	$2.5 \cdot 10^6$
		no	close-by	80	14 TeV	10^5
evaluation	$hh \rightarrow 4b$	yes	all	60	14 TeV	$9.2 \cdot 10^5$
		yes	close-by	60	14 TeV	$1.9 \cdot 10^4$
	$gg \rightarrow t\bar{t}$	yes	all	80	13 TeV	$2 \cdot 10^6$
		yes	close-by	80	13 TeV	$4.5 \cdot 10^4$

Table 5.2: Summary of all relevant datasets and their properties.

Chapter 6

Jet Reconstruction with Deep Neural Networks

In this chapter, two different approaches for jet reconstruction using deep neural networks are presented. Both are based on convolutional neural network architectures that receive two-dimensional calorimeter information (i.e. images) as inputs. The network of the first approach outputs a list containing all relevant jet information in an event, whereas the network's output of the second approach is an image itself. The employed architectures are fundamentally different, resulting in network-specific characteristics and different performances in reconstructing particle jets.

The goal of this chapter is the development of a network with the best possible jet reconstruction performance, regardless of its size and suitability in terms of a potential hardware implementation. An adaption to available hardware resources will be covered in Chapter 7.

6.1 First Approach: Sorted List of Jets

6.1.1 Basic Idea and Output Format

As already described in the previous sections, the inputs to the neural network will be grids of calorimeter trigger towers in the format of jFEX FPGA subregions, i.e. 24×32 towers in $\Delta\eta \times \Delta\phi$. Each FPGA subregion consists of a central region in which the jet finding algorithm is supposed to find jets and the environment, the overlap region with neighboring FPGAs. Thus, the goal is to train the network to reconstruct jets only in the FPGA core region while disregarding jets in the environment.

For this purpose, only information of anti- k_t jets whose center fall in the core region are given to the network as illustrated in Fig. 6.1 on the basis of two event displays. This approach has the additional advantage that the network will always observe only full jets shapes instead of also cropped jets at the edges and can thus focus to learn the geometry of these objects.

The desired output of the network corresponding to the input grid x_i will be a vector

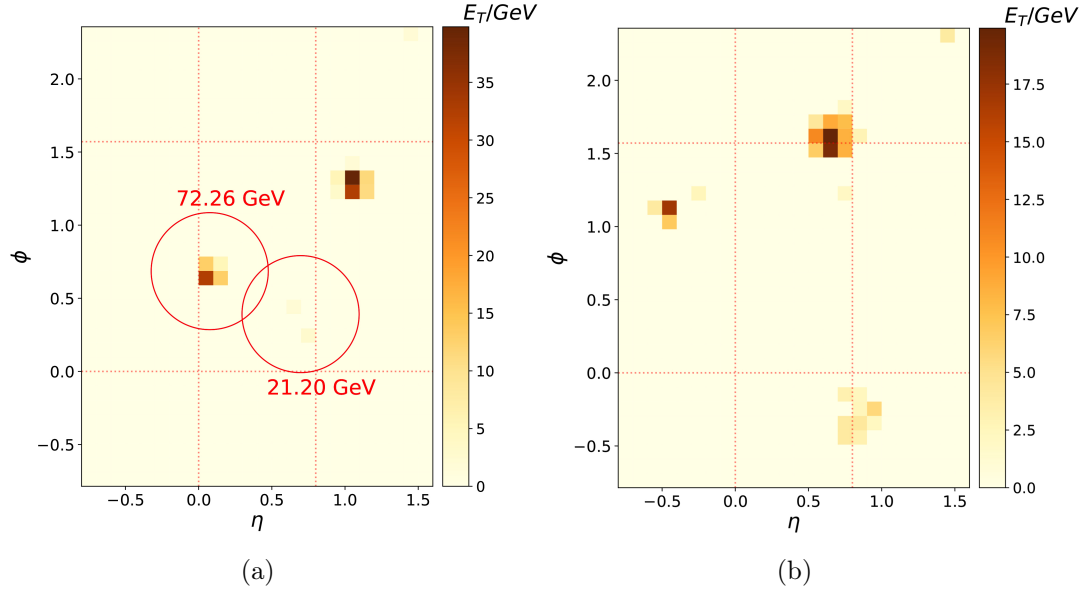


Figure 6.1: The network will be trained with anti- k_t jets in the core region (red). Jets in the environment are not supposed to be found.

$\vec{y}_i$, containing η , ϕ and E_T values of all jets in the core region. Additionally, we want the output to be sorted in E_T such that the jet with highest E_T comes in first place, followed by the jet with the second highest E_T and so on.

The static architecture of a neural network demands a fixed output vector. Given that the number of jets in an event is a dynamic quantity, the length of the output vector has to be chosen such that all jets will always be contained in the output. A reasonable maximal number of jets in the core region is $N_J = 10$. Hence, this will be used as the maximal number of jets in the output list.

Since each jet comes with three entries, the output vector will be of length 30 with the format

$$\vec{y}_i = (\eta_1, \phi_1, E_{T,1}, \eta_2, \phi_2, E_{T,2}, \dots, \eta_{N_J}, \phi_{N_J}, E_{T,N_J})^T, \quad (6.1)$$

and $E_{T,1} \geq E_{T,2} \geq \dots \geq E_{T,N_J}$ expressed in units of GeV.

If only $n < N_J$ jets are found (which will almost always be the case), the list is padded with zero, i.e. $(\eta_j, \phi_j, E_{T,j}) = (0, 0, 0) \forall n < j \leq N_J$.

6.1.2 Network Architecture and Hyperparameters

The chosen architecture for this problem is close to a CNN classification architecture as introduced in Section 4.2.2 but with some modifications: Instead of applying a logistic activation function in the output layer, ReLU activation functions are used for every layer in this regression network. Positional information is lost in pooling operations, therefore the network does not comprise pooling layers. It consists of an input layer followed by two convolutional layers, two fully-connected layers and finally

an output layer, as depicted in Fig. 6.2.

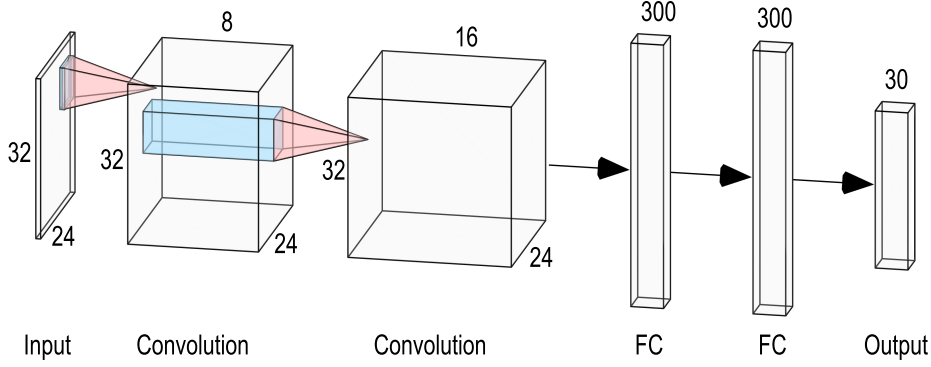


Figure 6.2: Jet reconstruction network consisting of convolutional and fully-connected (FC) layers. The depth of the convolutional layers represents their number of channels.

The first convolutional layer comprises 8 kernels with a size of 9×9 , the second one 16 kernels with the same size, while zero-padding is used to maintain the spatial extent of the input grid. The convolutional layers are followed by two fully-connected layers of 300 neurons each and the output layer with 30 neurons, the size of the output vector. This network has 3 887 370 trainable weights and will be trained with a batch size of $N_B = 256$. Dropout with $p = 0.3$ is used as a regularization method.

The mean squared error

$$E(X, \theta) = \frac{1}{2N_B} \sum_{i=1}^{N_B} \sum_{j=1}^{N_J} \left((\eta_j - \hat{\eta}_j)_i^2 + (\phi_j - \hat{\phi}_j)_i^2 + (E_{T,j} - \hat{E}_{T,j})_i^2 \right) \quad (6.2)$$

is used as the loss function without parameter norm penalty terms yet.

The input data is normalized onto the interval $[0, 1]$ employing the min-max normalization scheme with the tower energies $E_{T,\min} = 0 \text{ GeV}$ and $E_{T,\max} = 1100 \text{ GeV}$.

The features in the output vector will also be normalized onto the intervals $[0, 1]$ to ensure the same scale for every feature in the gradient and loss function. The minimal and maximal η and ϕ values are predetermined by the input size while the jet energies vary between $20 \text{ GeV} < E_{T,\text{jet}} < 2 \text{ TeV}$ for the $t\bar{t}$ training dataset.

After the network has been trained, the network's predictions will be rescaled to the original intervals for evaluation.

6.1.3 Simple Case Study: Single Jet Reconstruction

As a starting point the network is trained only on events containing exactly one single jet in the core region. This simple case study is conducted to make sure that no problems occur on a fundamental level and the network is able to find jets at all. For simplicity, fast simulation $t\bar{t}$ evaluation data is used in this section.

Fig. 6.3 shows the training and validation loss development during training.

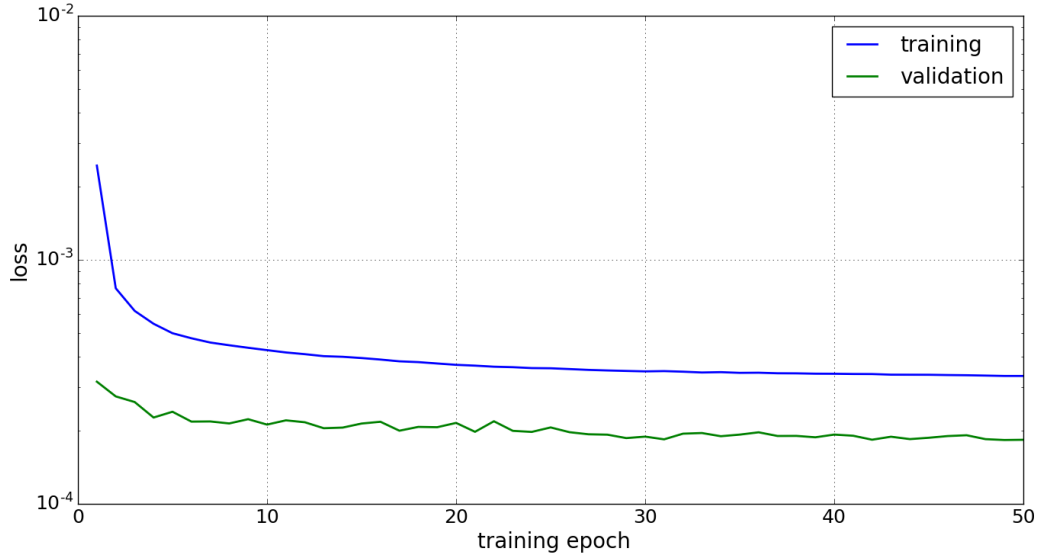


Figure 6.3: Training and validation loss development over the course of 50 training epochs for single jet reconstruction.

The training is stopped after 50 epochs where training and validation loss do not significantly decrease anymore. At all times, the validation loss is lower than the training loss which implies that the model performs well on unseen data and is thus able to generalize.

Fig. 6.4 shows two event displays with the jet location and energy predictions determined by the neural network and anti- k_t . As one can see, the network is capable of finding the single jet in the core region while disregarding all other jets in the environment. The position and energy predictions differ numerically from the anti- k_t values but not significantly.

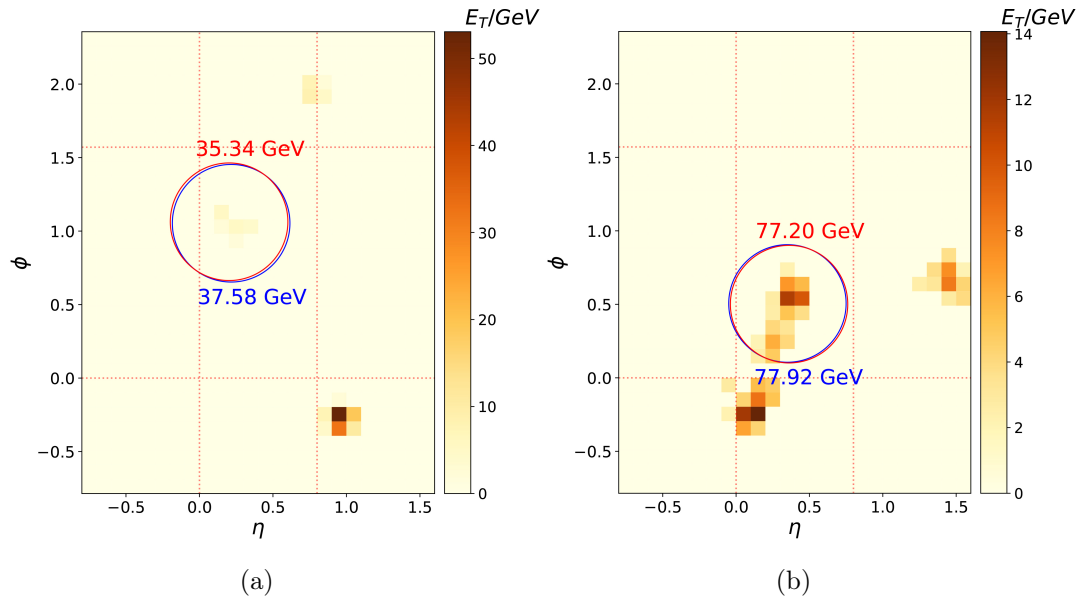


Figure 6.4: Example event displays with one single jet in the core region from the $t\bar{t}$ evaluation dataset together with the locations and energies determined by the neural network (blue) and anti- k_t (red).

Before further analysis of the network's performance, it can be advantageous to understand what the network has learned on a basic level. For this purpose the eight kernels of the first convolutional layer of the network are illustrated in Fig. 6.5. Some kernels seem to be sensitive to certain round shapes with a special focus on kernel 5 which nicely represents the shape of a jet with radius $R \approx 0.4$ as prescribed by the anti- k_t metric. But also kernels 3 and 2 have learned to detect round features. For these kernels, central regions contribute positively to the total jet energy and outer regions act as some kind of overlap removal and deduct energy depositions from the overall energy. Even though there is only one single jet in the core region, these jets can already overlap with jets in the environment what is taken into account by these kernels. The network has thus learned to incorporate certain energies for determining the jet's energy while subtracting or ignoring other energy depositions that are not relevant for the reconstruction. Especially kernel 6 functions as a counterpart to the jet finding kernels above, supposedly focusing on the jets in the environment that need to be disregarded.

Other kernels seem to contain random noise. Noisy kernels generally do not have a large influence on the outcome but will become important as the problem's complexity increases with a variable number of jets. Multiple (possibly overlapping) jets require more complex features that the network is able to detect and thus these kernels will be trained to be sensitive to these new features.

The kernels in the second layer represent higher-level features which cannot be interpreted as easily as the kernels in the previous layer (see Appendix A.1).

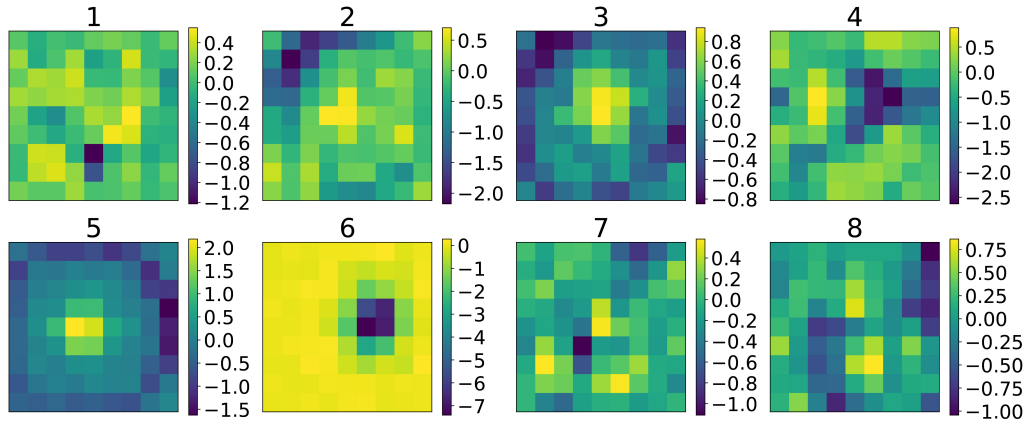


Figure 6.5: Illustration of the eight 9×9 kernels in the first convolutional layer of the single jet finding network.

Figures 6.6 and 6.7 show the neural network's angular and energy resolutions with respect to anti- k_t truth together with the resolution distributions for anti- k_t online. In order to compare the following results quantitatively, the distributions are fitted using a Gaussian fit function, which will give a rough estimate on the algorithm's performance, with three fit parameters p_0 (scaling factor), μ (mean value) and σ (standard deviation): $f(x) = p_0 \cdot \exp(-0.5((x - \mu)/\sigma)^2)$. For reasons of clarity, the fit functions are not depicted in the figures. Table 6.1 summarizes all fit parameters.

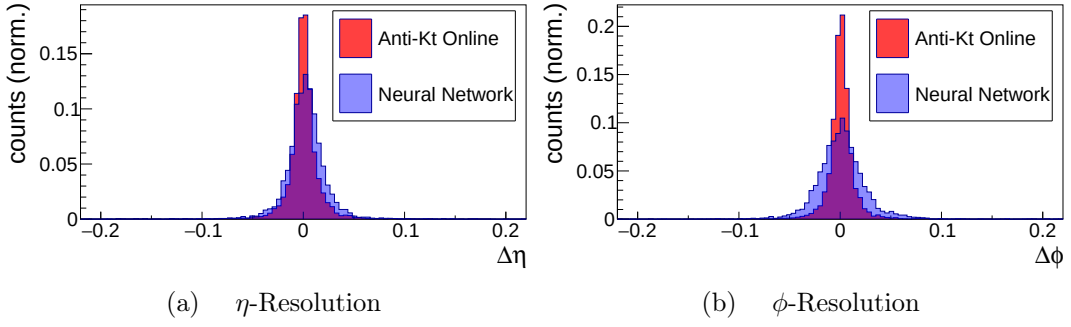


Figure 6.6: Angular resolution in η and ϕ for the neural network in comparison with anti- k_t online, defined as the difference to the anti- k_t truth values: $\Delta\eta = (\eta_{\text{truth}} - \eta)$ and $\Delta\phi = (\phi_{\text{truth}} - \phi)$.

The network's η and ϕ resolutions are relatively close to the anti- k_t online resolutions. However, the neural network energy distribution in Fig. 6.7 is much worse than the anti- k_t online one and slightly asymmetric (mean: 0.03) in such a way that the network tends to predict smaller energy values in comparison to the true ones. The poor resolution especially of the jet energy prediction results from the necessary energy scaling in the output list. Here, the entire jet energy range ($20 \text{ GeV} < E_T < 2 \text{ TeV}$) is squeezed into the interval $[0, 1]$ in order to guarantee the same gradient scale for all output features. During training, the loss function ensures that all features have the same accuracy within the interval $[0, 1]$. However, the subsequent rescaling to the original interval worsens the resolution especially for large range features like the jet energy. The same effect is visible in Fig. 6.6 for the resolution in ϕ , which is worse than the one in η since the ϕ interval ($0 \leq \phi \leq \pi/2$) is almost twice as large as the interval in η ($0 \leq \eta \leq 0.8$).

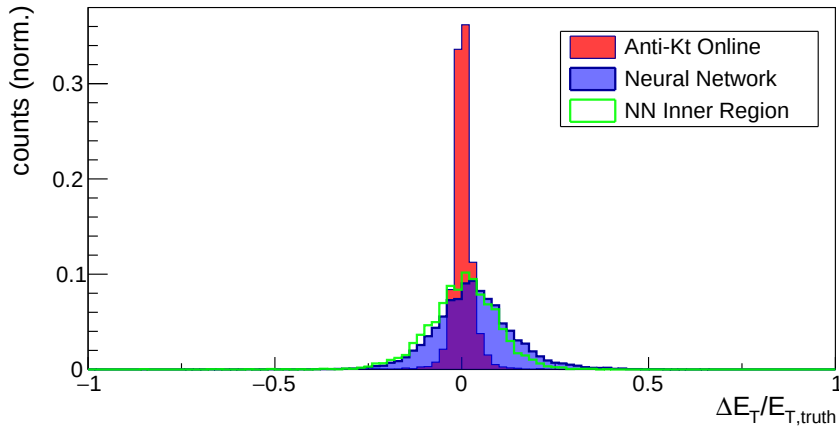


Figure 6.7: Jet energy resolution for the neural network and neural network in innermost region of the core region ($0.2 < \eta < 0.6$ and $1/8\pi < \phi < 3/8\pi$) in comparison with anti- k_t online, defined as the difference to the anti- k_t truth values: $\Delta E_T = E_{T,\text{truth}} - E_T$. The jet energies vary between $20 \text{ GeV} < E_{T,\text{truth}} < 2 \text{ TeV}$.

The energy resolution asymmetry arises from forcing the network to predict jets only

in the core region in combination with zero energy entries in the output list for not existing jets: For jets close to the environment boundary, smaller jet energy values are more favorable for minimizing the loss function since accidentally predicted jets with large energy values would result in a large penalty for the loss function. Fig. 6.7 shows also the energy resolution for jets in the innermost region of the core region ($0.2 < \eta < 0.6$ and $1/8\pi < \phi < 3/8\pi$) where these boundary effects are not present anymore and the distribution is symmetric with a mean value of about 0.002.

Distribution	Algorithm	$p_0/10^{-1}$	$\mu/10^{-3}$	$\sigma/10^{-2}$
$\Delta\eta$	anti- k_t online	1.53 ± 0.01	3.06 ± 0.18	0.95 ± 0.01
	neural network	1.19 ± 0.02	8.15 ± 0.07	1.38 ± 0.22
$\Delta\phi$	anti- k_t online	1.78 ± 0.03	9.27 ± 0.21	0.82 ± 0.01
	neural network	1.02 ± 0.02	14.06 ± 0.53	1.66 ± 0.24
$\Delta E_T/E_T$	anti- k_t online	3.75 ± 0.74	1.87 ± 0.23	1.97 ± 0.02
	neural network	0.62 ± 0.12	34.23 ± 2.18	9.12 ± 0.42
	NN inner region	0.76 ± 0.14	2.36 ± 1.38	8.82 ± 0.28

Table 6.1: Summary of fit parameters p_0 (scaling factor), μ (mean value) and σ (standard deviation) for angular and energy resolution fits.

A similar effect is observable for the η and ϕ predictions of jets close to the environment boundary. Here, the forced cut introduces a bias in such a way that the network tends to shift the jet positions away from the boundary. Generally, these jets are very risky for the network and its loss function and therefore jets further away from the boundary are preferred. Fig. 6.8 shows η and ϕ projections of the ratio of jets found by the neural network and jets found by anti- k_t truth. The network does not predict any jets in the environment region and thus executes the cut perfectly. However, the lag of jets in the outer FPGA core region bins together with an enrichment in the neighboring inner bins clearly shows the bias effect described above.

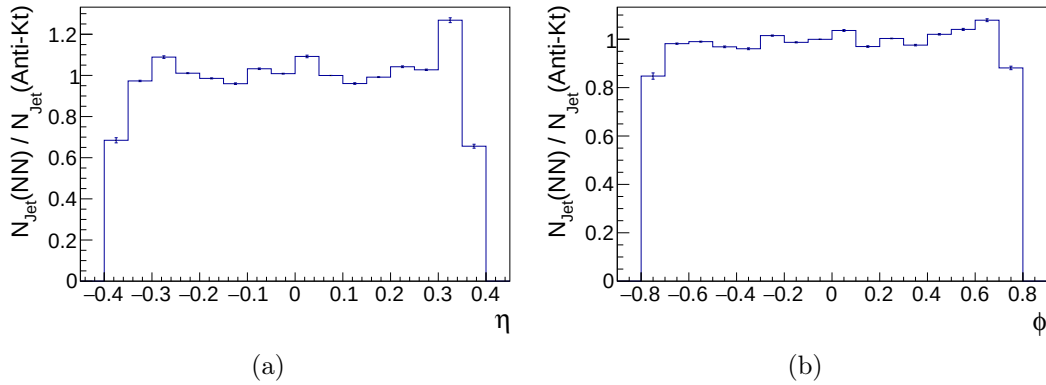


Figure 6.8: Ratio of jets found by the neural network and jets found by anti- k_t truth, projected in (a) η and (b) ϕ .

Fig. 6.9 shows the overall single jet finding efficiency curves (also known as “turn-on curves”) of the network and anti- k_t online with respect to anti- k_t truth. A perfect efficiency would result in a step function at the chosen jet energy threshold $E_T^{\text{cut}} = 100$ GeV, meaning that all true jets above the threshold were found while no jets below the threshold were accidentally reconstructed with an energy above E_T^{cut} . A direct consequence is that the slope of the turn-on curve represents the energy resolution of the algorithm. As one can see, the slope of the network’s efficiency curve is less steep than the anti- k_t online one which was to be expected due to the differences in energy resolution. However, the neural network reaches the plateau with a jet finding efficiency of 100% almost as fast as anti- k_t online does. Hence, despite some general boundary problems, the network is able to find one single jet with an efficiency close to the one of anti- k_t online.

Note that these efficiency curves are only used to demonstrate the slopes and plateaus reached by the algorithms and do not comprise any matching to trigger rates yet.

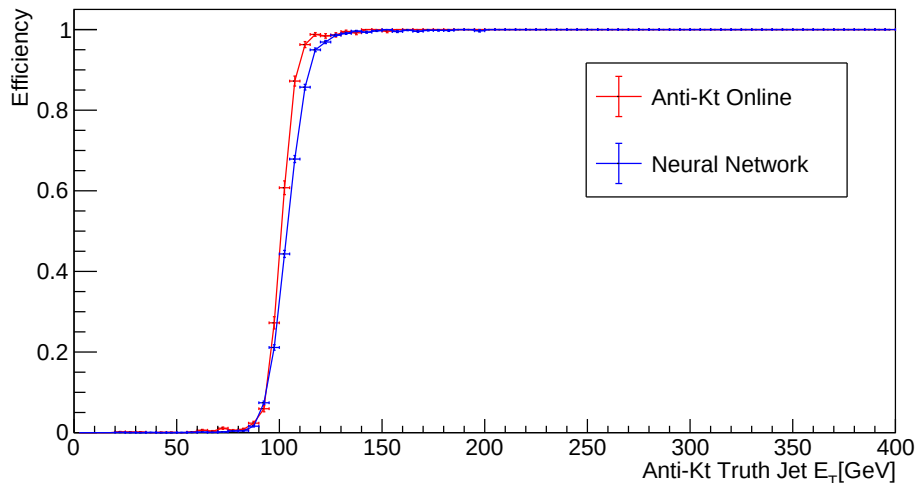


Figure 6.9: Single jet finding efficiency curves for the neural network and anti- k_t online with respect to anti- k_t truth jets for jet energies of $E_T > 100$ GeV.

6.1.4 Multiple Jet Reconstruction

The same neural network architecture from the previous section is now used to reconstruct a variable number of jets (≤ 10), representing the most general case. This study is again conducted on fast simulation $t\bar{t}$ data without pile-up. The network is trained over 50 epochs with the same set of hyperparameters. The goal remains finding jets only in the FPGA core region while now the jet energy sorting of the output comes into play. While the output vector was always filled with zeros except for the first three entries in the previous case study, the network has now to predict the correct number of jets in the core region and fill the output vector accordingly.

Fig. 6.10 shows two event displays with different numbers of jets together with the

neural network and anti- k_t truth predictions. In these cases, the network is capable of finding a variable number of jets while assigning reasonable energy values to the respective jets, even if they are overlapping ($\Delta R < 0.8$) like in Fig. 6.10(b).

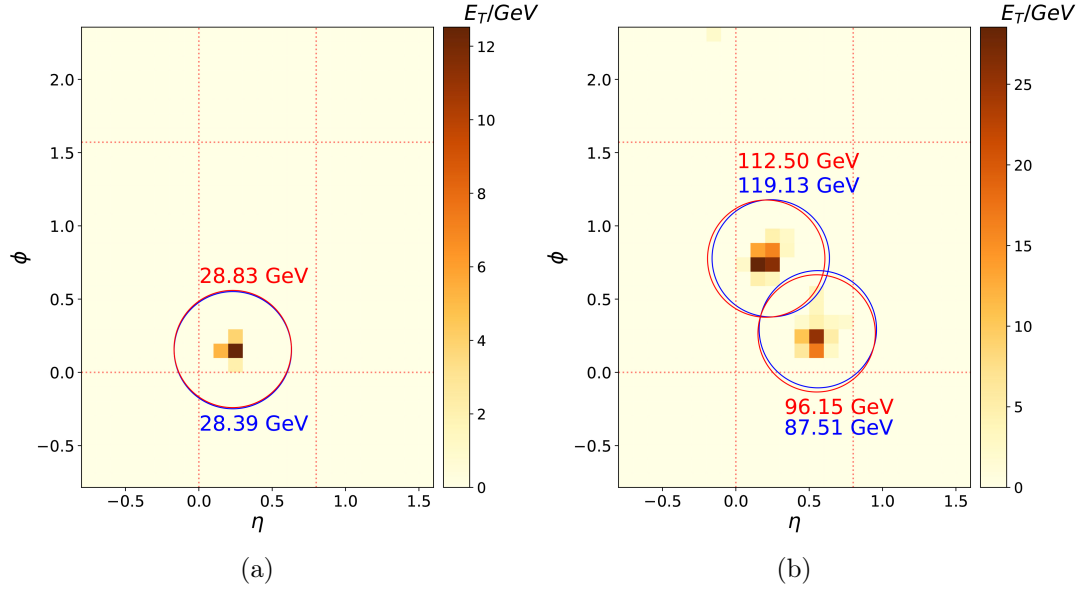


Figure 6.10: Example event displays for a variable number of jets in the core region from the fast-simulated $t\bar{t}$ evaluation dataset together with the locations and energies determined by the neural network (blue) and anti- k_t (red).

The demanded sorting of the output list in E_T works perfectly well: All jets are always (in exactly 100% of all cases) sorted such that the jet with the highest *predicted* energy value comes first, followed by the jet with the second highest E_T and so on. This network intrinsic sorting does of course not necessarily reflect the true order of jet energy values due to the finite energy resolution of the network, as shown in Fig. 6.12. If several jets within one event have close energy values, the order might differ from the order given by anti- k_t .

Table 6.2 shows the fit parameters obtained by fitting the angular resolution distributions in Fig. 6.11 and energy resolution in Fig. 6.12.

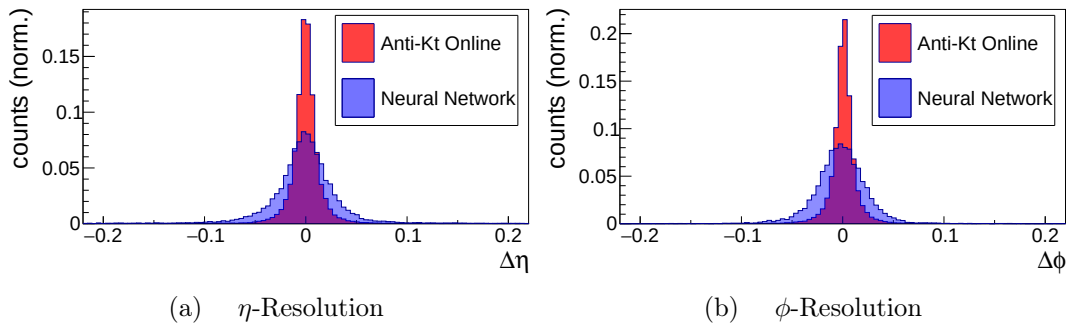


Figure 6.11: Angular resolution in η and ϕ for the multi-jet finding neural network in comparison with anti- k_t online, defined as the difference to the anti- k_t truth values: $\Delta\eta = (\eta_{\text{truth}} - \eta)$ and $\Delta\phi = (\phi_{\text{truth}} - \phi)$.

The angular resolutions in η and ϕ worsened in comparison to the single jet finding case in the previous section. Also the network's energy resolution for the general case is worse and more asymmetric than for the single jet scenario. Note that the anti- k_t online resolution also worsened for this considerably more complex task. Again, the network's accuracy in the innermost region of the FPGA subregion is more symmetric and more comparable to the anti- k_t online resolution than in the previous simple case study.

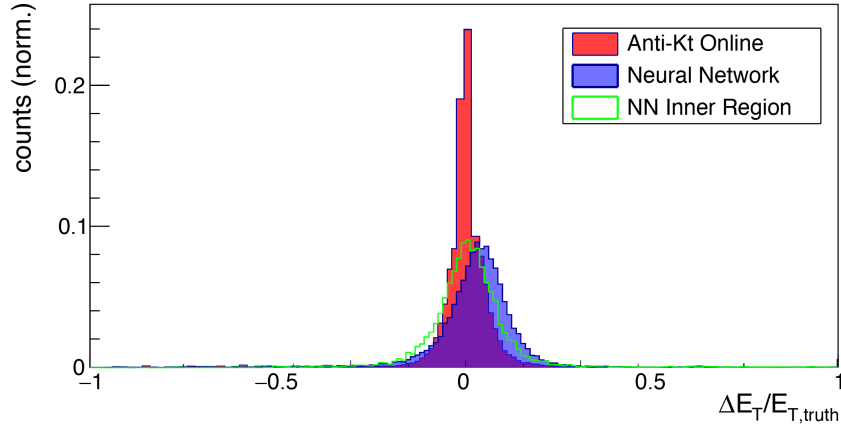


Figure 6.12: Jet energy resolution for the multi-jet finding neural network and neural network in innermost region of the core region ($0.2 < \eta < 0.6$ and $1/8\pi < \phi < 3/8\pi$) in comparison with anti- k_t online, defined as the difference to the anti- k_t truth values: $\Delta E_T = E_{T,\text{truth}} - E_T$. The jet energies vary between $20 \text{ GeV} < E_{T,\text{truth}} < 2 \text{ TeV}$.

Distribution	Algorithm	$p_0/10^{-1}$	$\mu/10^{-3}$	$\sigma/10^{-2}$
$\Delta\eta$	anti- k_t online	1.52 ± 0.01	-0.02 ± 0.01	0.97 ± 0.01
	neural network	0.71 ± 0.11	0.95 ± 0.04	2.24 ± 0.21
$\Delta\phi$	anti- k_t online	1.75 ± 0.02	0.95 ± 0.04	0.83 ± 0.01
	neural network	0.46 ± 0.09	-9.47 ± 0.43	3.36 ± 0.41
$\Delta E_T/E_T$	anti- k_t online	2.12 ± 0.18	6.67 ± 0.38	3.45 ± 0.04
	neural network	0.55 ± 0.04	58.58 ± 0.89	11.69 ± 0.12
	NN inner region	0.69 ± 0.05	7.26 ± 0.68	8.72 ± 0.09

Table 6.2: Summary of Gaussian fit parameters p_0 (scaling factor), μ (mean value) and σ (standard deviation) for angular and energy resolution of Fig. 6.11 and 6.12.

The worse resolutions especially in η and ϕ in comparison to the previous single jet finding case mostly arise from a training artifact, explained in the following.

Fig. 6.13 shows two events with two anti- k_t truth jets each. If the true jet energies

are close to each other and the two jets therefore look very similar to the network, the positions of the jets are predicted incorrectly, in fact the network moves the positional prediction close to the common center of both true jets. In this case, also the network's energy predictions differ significantly from the true values.

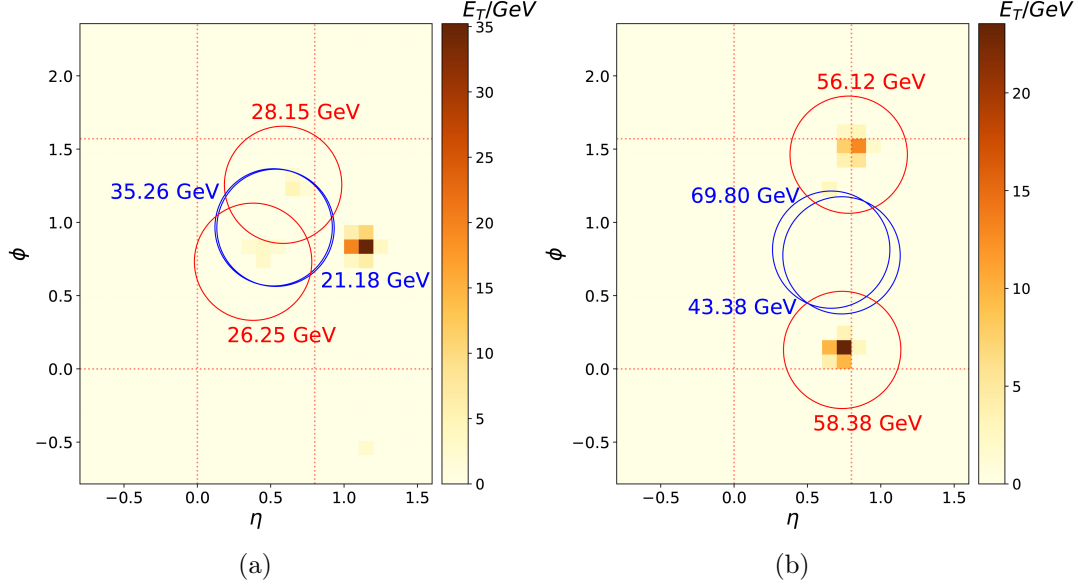


Figure 6.13: Example event displays for events with two anti- k_t jets (red) with close energies. The neural network (blue) fails to predict the correct positions and places both jets in the middle of the truth jets instead.

This artifact can be traced back to the training process: Consider the situation when training the network on the event displayed in Fig. 6.13(b). In this case, the position prediction of the first jet in the output list (the higher-energetic true jet) is moved downwards while the prediction of the second jet is moved upwards. Since the training dataset is very large, the network will come across a similarly looking event during training, but this time the upper jet in the event might be the higher-energetic one. Thus, the position prediction of the first jet in the output list is now moved upwards and the one of the second jet downwards — exactly the opposite of the previous update step. Since the network will encounter hundreds or even thousands of these cases, the η and ϕ predictions end up in the averaged center between the two anti- k_t jets.

Fig. 6.14 shows the ratio of events with two incorrectly predicted neural network jets and all events featuring two jets for true jet energy differences up to 50 GeV.

Even though this incorrect prediction occurs especially if the jets look very similar to the network, which is most often the case if the energy difference between the two jets is small, it also occasionally occurs for higher energy differences. The exact same artifact can also be observed for events with more than two jets. Since already events with three jets in the FPGA core region make up only 0.03%, events with more than two jets were not taken into account here.

A potential solution to this problem could be an algorithm that flips the jets in

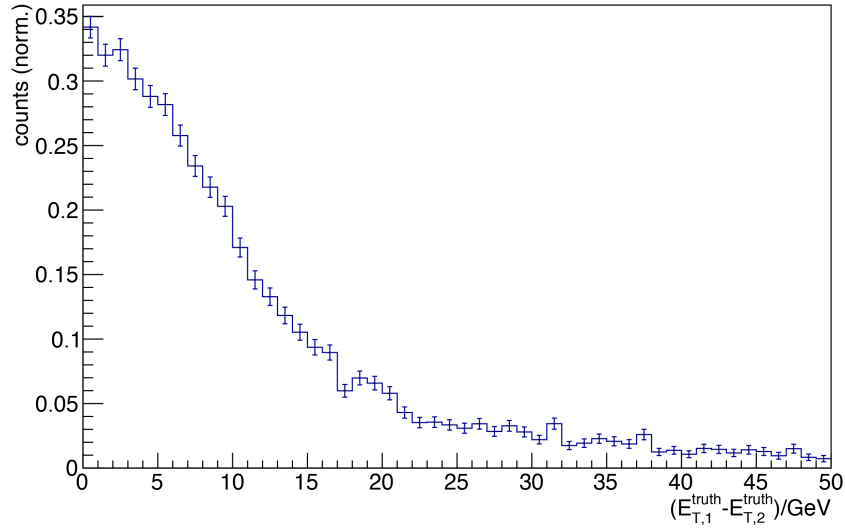


Figure 6.14: Ratio of events with two neural network jets incorrectly shifted to the center and all true two jet events.

the target vector during training every time the network tries to move the angular prediction in the wrong direction. In order to do this, one could calculate the mean squared error for the current target vector order and also for a flipped version of it at each training step. If the error is smaller for the flipped version, the target vector is changed accordingly.

Besides sacrificing the energy ordering, the solution is computationally extremely expensive, especially if one considers an arbitrary amount of jets to be flipped in the output vector.

Fig. 6.15 shows the efficiency curves for the multi-jet finding neural network and anti- k_t online. Due to the broad energy resolution, the slope of the network's turn-on

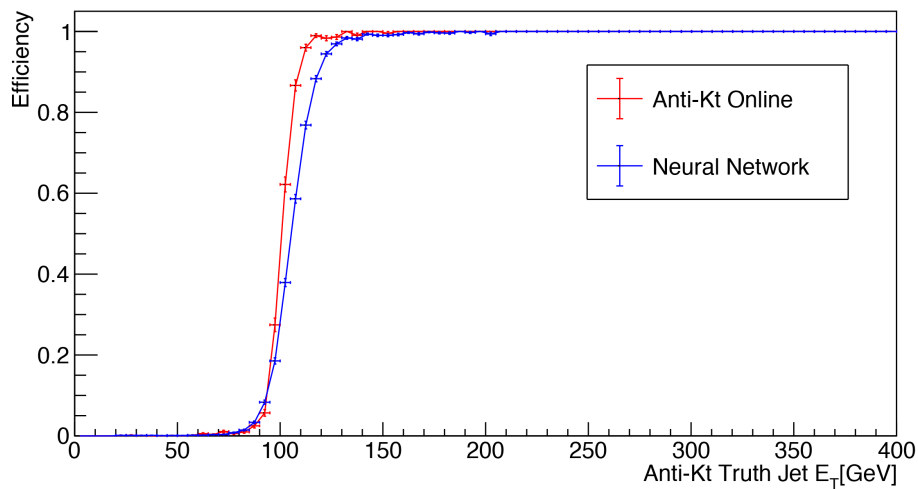


Figure 6.15: Multi jet finding efficiency curves for the neural network and anti- k_t online with respect to anti- k_t truth jets for jet energies of $E_T > 100$ GeV.

curve is less steep compared to anti- k_t online. Also the plateau is only fully reached at about 200 GeV, which is worse in comparison to the previous single jet finding case. In spite of having the presented problems, the network's efficiency is still good, even if not comparable to the one of anti- k_t online.

6.1.5 Summary of Presented Approach

The neural network presented in this section is capable of finding jets. The demanded cut at the FPGA core region borders is executed perfectly but introduces biases especially for jets close to these boundaries. These biases arise due to the necessity of padding the output list with zeros for not present jets since the static architecture of a neural network requires a fixed output list. Due to the scaling of the output vector, the resolution of large scale features like the jet energy decreases. Additionally, having the jets ordered in energy leads to incorrectly predicted jets which are moved to the common center of several true jets in the case of multi-jet reconstruction.

Despite the presented issues of the network, the efficiency curves reach a plateau and exhibit a relatively steep slope, even if not reaching the anti- k_t online efficiency.

By discarding the core region cut-off and implementing a (potentially computationally very expensive) flipping algorithm, the efficiency and accuracy of the network could certainly be improved. However, these partly fundamental problems and also the necessary network size with millions of trainable parameters, which will be problematic with regard to a potential FPGA implementation, lead to the conclusion that another approach and network architecture might be beneficial for our purposes.

6.2 Second Approach: Regression on Grid

6.2.1 Basic Idea and Output Format

The second approach presented in this section makes use of a different output format and thus an entirely different network architecture. While the input to the neural network will again be a grid of calorimeter trigger towers in the format of jFEX FPGA subregions, the output will now be a grid of the same size as the input, i.e. a grid with dimensions 24×32 towers in $\Delta\eta \times \Delta\phi$.

For every jet that has been found by the algorithm, the corresponding tower in the output grid which contains the jet center will be filled with the entire jet energy. This means that the target output grid for training will represent all necessary anti- k_t truth information in the form of single pixel values, as illustrated in Fig. 6.16.

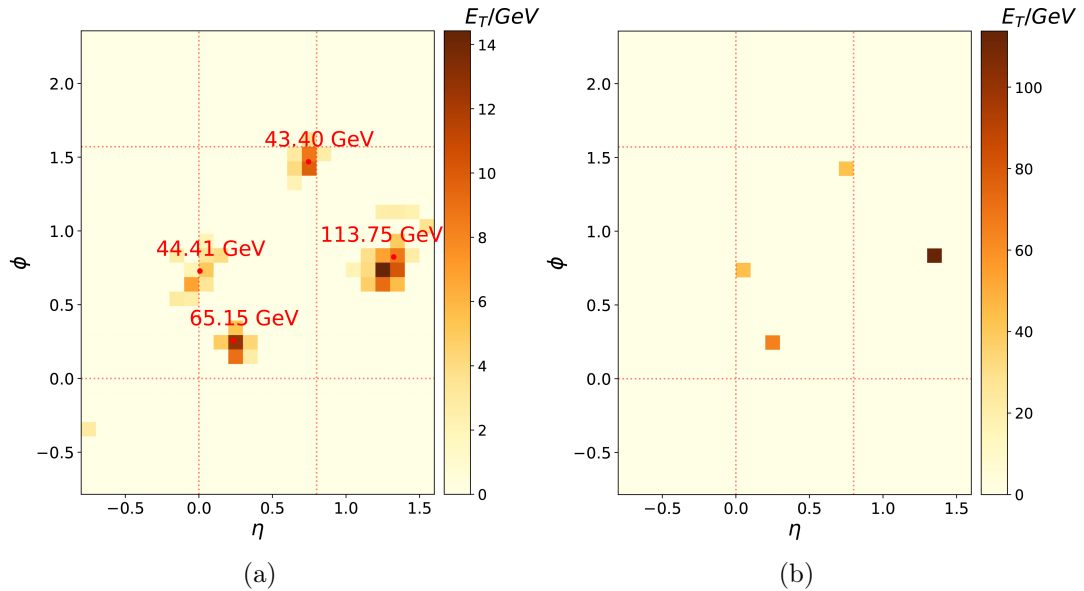


Figure 6.16: (a) Input grid of calorimeter trigger towers with four jets found by anti- k_t truth. The jet center positions are depicted by red dots. (b) The corresponding target output has the same size as the input and is filled with pixels at the positions of the respective jet centers containing the full jet energy.

The format of the output has been chosen in such a way that the problems of the previously presented approach cannot occur here by construction:

Firstly, a variable number of jets can be predicted without the need of padding zeros for not present jets. Also, no direct maximum number of jets is predetermined by this format.

Secondly, since the angular predictions are implicitly contained in the positions of the filled pixels, the only feature scale in the output is the scale of the energy. Therefore, a normalization of the output grid is not necessary and the network can directly predict the total jet energy values which should lead to better energy resolutions. Of course, the discretization of η and ϕ artificially worsens the angular resolution, however, an

accuracy of 0.1×0.1 in $\Delta\eta \times \Delta\phi$ is sufficient for a trigger jet finding algorithm.

Lastly, the jets do not need to be ordered in a certain way, such that training artifacts like the one identified in Section 6.1.4 cannot occur.

Since a core region selection cut learned by the network could again introduce biases, the network will be trained to reconstruct all jets in the entire FPGA subregion, including the environment. A selection of jets in the core region will be applied afterwards.

6.2.2 Network Architecture and Hyperparameters

Since the network's task is to create a new grid featuring jet information, a network architecture that is capable of creating images is needed. Section 4.2.2 introduced the concept of image segmentation architectures that can also be employed as convolutional auto-encoders for (re-)constructing images. Fig. 6.17 shows the architecture of the jet finding network used in this second approach. It is based on the structure of a convolutional auto-encoder, as described in Section 4.2.2. Additionally, layers with the same size from both paths can be concatenated, resulting in a U-net-like architecture [30]. This can be advantageous as important information can skip down-sampling operations and thus directly flow to the up-sampling path. Also weights can be updated more easily during training at the beginning of the network, since the gradient can be propagated through these concatenations.

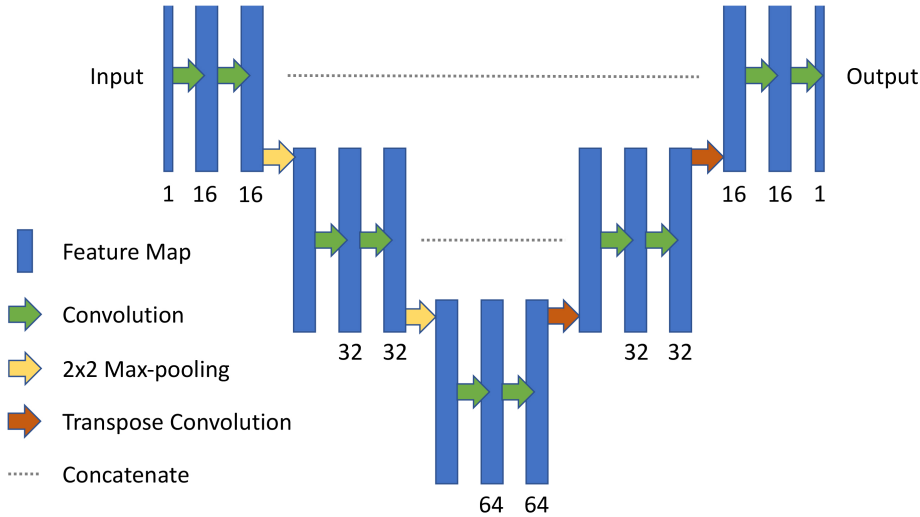


Figure 6.17: Illustration of the network architecture, consisting of a down-sampling path (left) which involves convolution and 2×2 max-pooling and an up-sampling path (right) which involves 2×2 transpose convolution and convolution.

The input layer is followed by two convolutional layers with 16 kernels each. The two convolutional layers after the first max-pooling operation feature 32 kernels each and are followed by another max-pooling and in turn two convolutional layers, both featuring 64 kernels. All kernels come with size 3×3 as they gave the best performance.

Full jet shapes will still be observed by these kernels in the down-sampled feature maps. The convolutional layers in the up-sampling path come with the same parameters as their down-sampling counterparts. The up-sampling steps are performed using transpose convolution, i.e. a learnable up-sampling, with size 2×2 and stride 2. The first up-sampling layer comprises 32 transpose convolution filters, the second one 16. In order to obtain a two-dimensional output grid, i.e. a single-channel image, only one single kernel of size 3×3 is used in the very last convolutional layer. The ReLU activation function is used for all layers throughout the network.

As already mentioned above, the network's output will not be normalized, i.e. the network will directly output the jet energy values. Given that the network has to deal with large output numbers, it is beneficial to leave the input data unnormalized as well so that values of the same scale are used throughout the network.

The network has 116 881 trainable weights, a substantial reduction compared to the previous approach, and will be trained with a batch size of $N_B = 256$. Again, dropout with $p = 0.3$ is used as a regularization method and the mean squared error, understood to be applied pixelwise, as a loss function.

6.2.3 Performance Analysis

The network is trained on close-by $t\bar{t}$ data without pile-up. Events containing close-by jets are chosen to make the network more sensitive to the challenging reconstruction of overlapping jets. The training was stopped after 50 epochs when the validation loss did not significantly decrease anymore. Fig. 6.18 shows the development of training and validation loss during training. As one can see, this network is also able to generalize and performs well on unseen validation data.

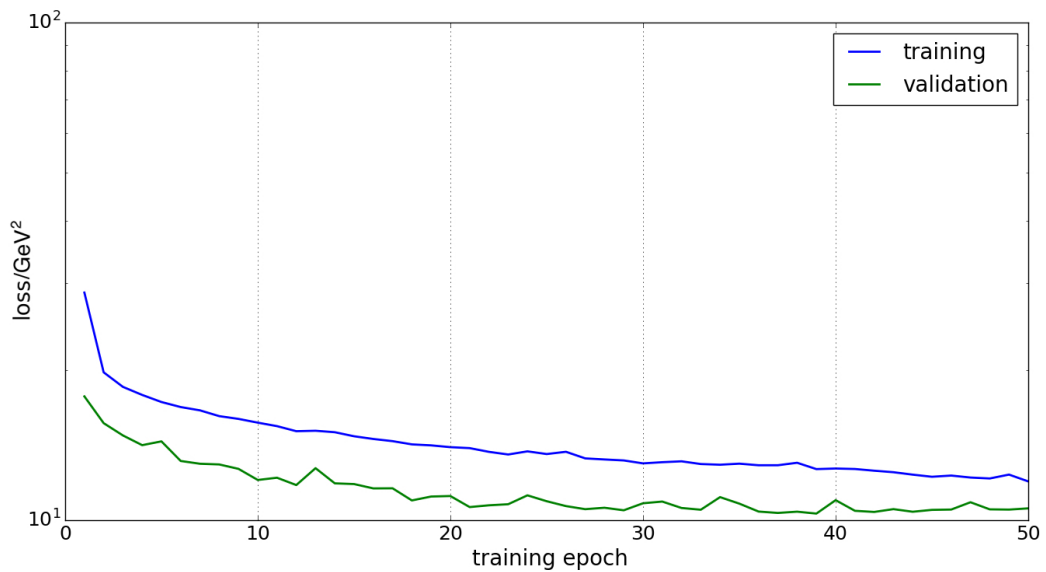


Figure 6.18: Development of training and validation loss over the course of 50 training epochs.

Since the output of the network is not normalized, the validation loss in Fig. 6.18 can directly be used to estimate how well the network performs on new data: the mean squared error over all tower entries will be of the order of 10 GeV^2 .

The 16 kernels with size 3×3 of the first convolutional layer are depicted in Fig. 6.19. Since these kernels are too small to contain full jet shapes or circular features, a direct interpretation of their functions is not possible anymore. Note, however, that – despite using no input normalization – all weights are small with the majority being very close to zero which is desired to ensure the same gradient scale during training. This is also visible in Fig. 6.20 which shows the weight distribution of all trained network weights (including biases).

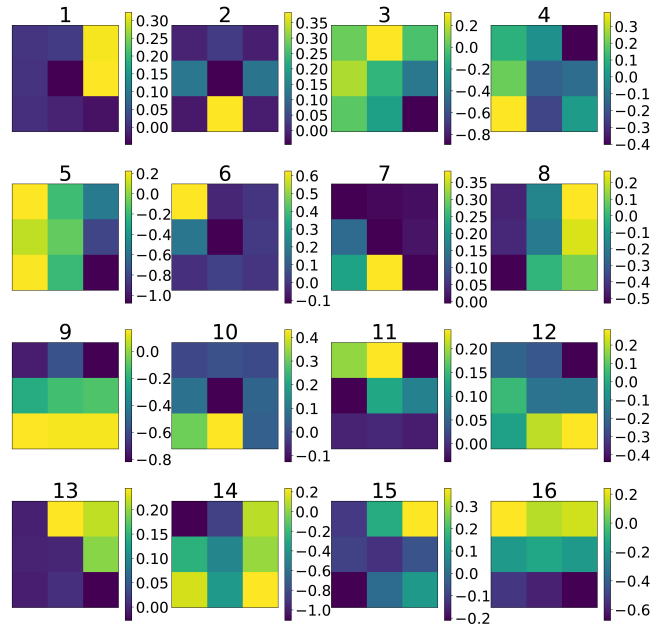


Figure 6.19: Illustration of the 3×3 kernels of the first convolutional layer of the network.

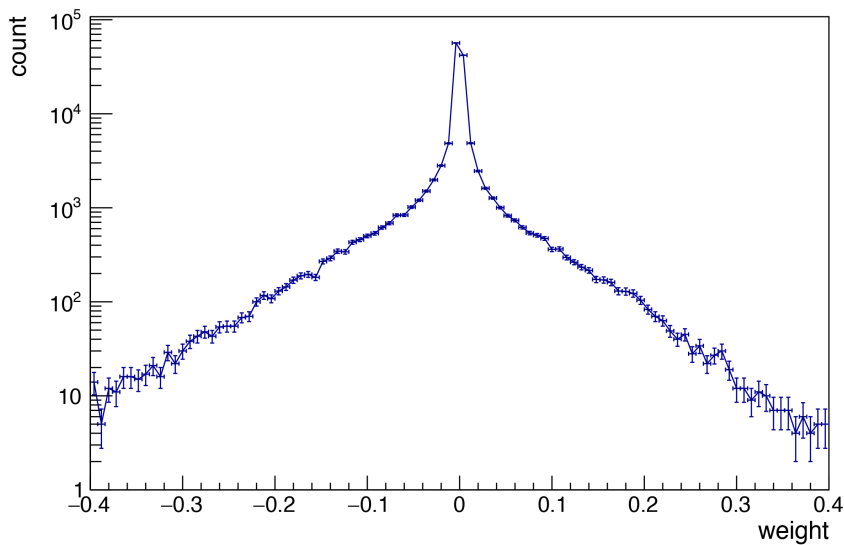


Figure 6.20: Weight distribution of all 116 881 weights and biases.

In order to better understand what the network has learned during training one can analyze the influence of towers surrounding a jet on its total jet energy. For this purpose a single fast-simulated 200 GeV jet is placed at a fixed position in the FPGA core region. Every single tower in the entire FPGA subregion is then separately increased by an energy $E_c = 10$ GeV at a time, leading to $24 \times 32 = 768$ different tower grids, all containing the same jet and only differing in exactly one single tower entry. The neural network is applied on all 768 grids to predict the jet's energy which is expected to be higher if the tower with the increased energy is close to the jet position and unchanged for grids where the increased tower is far away from the jet. The relative influence of the increased tower energy on the entire jet energy is recorded for every single tower position in the FPGA subregion. This procedure is repeated 50 times for different jet shapes at the same jet position while averaging over all relative tower influences. Fig. 6.21 shows the influence regions for two different jet positions.

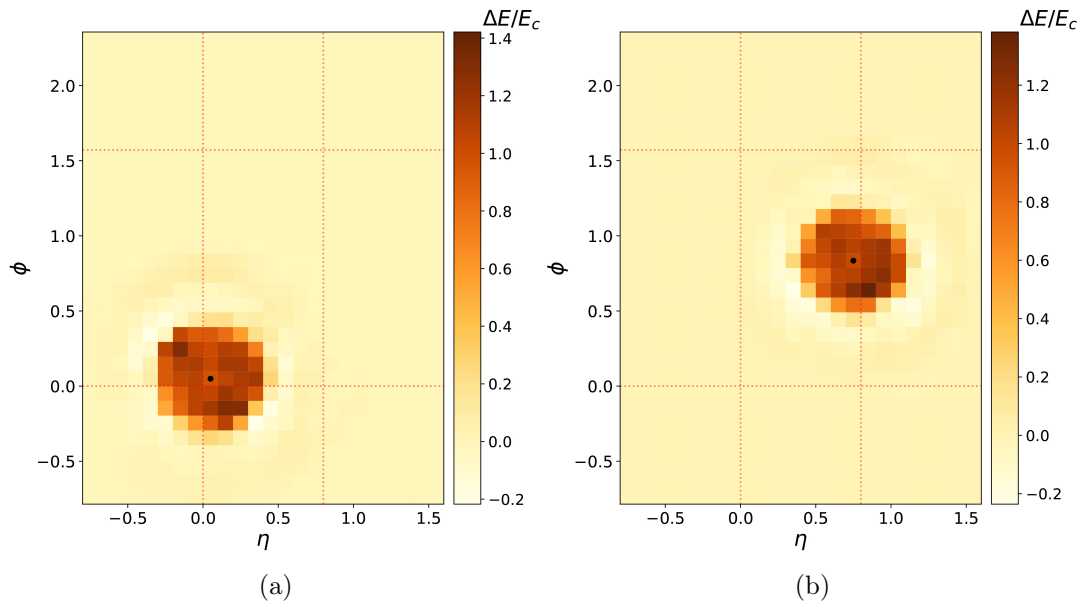


Figure 6.21: Influence regions for two different jet positions (black dots), averaged over 50 different jet shapes each. The energy differences ΔE between the original jet energy of 200 GeV and the newly determined once relative to the altered tower energy E_c are shown for the respective tower positions.

As one can see, the network has perfectly learned to determine the total jet energy by taking towers within a radius of $R = 0.4$ into account as prescribed by the anti- k_t metric. The regions are overall symmetric, containing only small fluctuations. For distances slightly greater than R , the relative tower influence on the total jet energy is negative, meaning that positive tower energies reduce the jet's energy which is a manifestation of the learned overlap removal for close-by jets. Lastly, towers far away from the jet center do not contribute to the total jet energy as expected.

Influence regions for different jet positions look similar to the ones above (see A.2), implying that the network's behavior does not depend on the jet position which is desired for a universal jet reconstruction.

Fig. 6.22(a) shows an event of the $t\bar{t}$ fast simulation evaluation dataset together with the positions and energies of the anti- k_t truth jets. The corresponding output of the neural network is illustrated in Fig. 6.22(b). All anti- k_t truth jets have been found

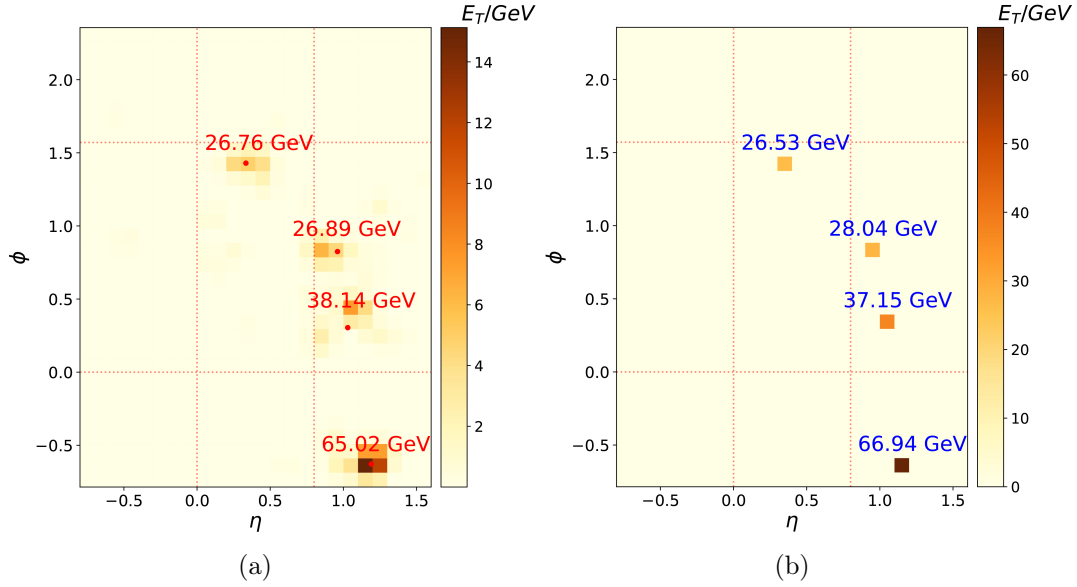


Figure 6.22: (a) Example event display ($t\bar{t}$ fast simulation) with anti- k_t truth jet positions and energies. The red dots mark the locations of the jet centers. (b) Jets found by the neural network.

by the neural network with jet energies close to the true ones. The network has also predicted the exact towers in which the anti- k_t jet centers fall. Problems that occurred in the previous approach such as the shifting of two jet position predictions to the middle of the two jet with close energy values are not existent.

Multiple Pixel Prediction – Training Artifact

However, also this approach comes with an artifact introduced during training. Instead of filling the output grid with exactly one single pixel for each jet, the network fills also multiple neighboring pixels at a time, which is demonstrated in Fig. 6.23 for one example $t\bar{t}$ event.

This behavior only occurs if the jet center is close to one of the boundaries of its tower and therefore close to a neighboring tower. During training, the network observes a lot of jets whose centers are almost exactly at the edge between two towers. In some cases the network has to fill the first one while leaving the second one empty, in other cases it will be exactly vice versa. Since the penalty to the loss function will be huge if the network accidentally fills a large jet energy value into a tower that is supposed to be empty (or predicts an empty tower where the target grid contains a large energy), filling both towers with nonzero values that roughly add up to the total energy is preferred to minimize the loss function.

Roughly 58.2% of all jet predictions comprise more than one output tower.

Since there is a minimum distance of $\Delta R = 0.4$ between two jets (otherwise they

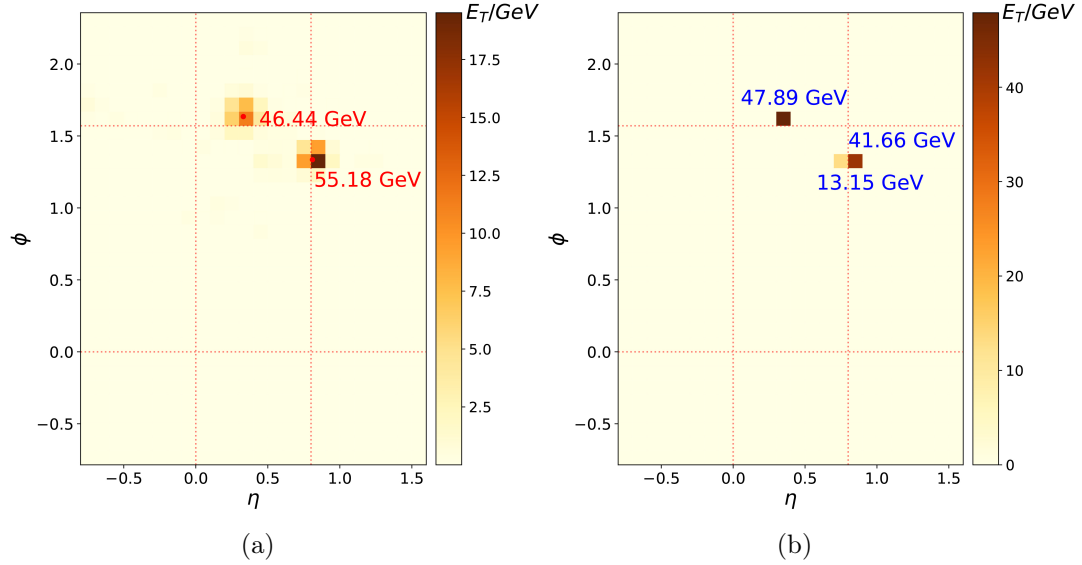


Figure 6.23: (a) Example $t\bar{t}$ event with two anti- k_t jets (red). (b) The neural network fails to output a single tower for the higher-energetic jet and fills the output grid with multiple neighboring pixels instead (network predictions given in blue).

would have been merged by anti- k_t), directly neighboring pixels cannot be two individual jets and can therefore be summed up to one pixel, representing one single jet. If multiple neighboring pixels are predicted by the neural network, the pixel containing the local maximum is filled with the sum of a 3×3 grid, centered around that maximum, while the remaining eight pixels are set to zero. Fig. 6.24(a) illustrates this summation procedure, resulting in a grid containing only single pixels, each of which representing one jet (Fig. 6.24(b)).

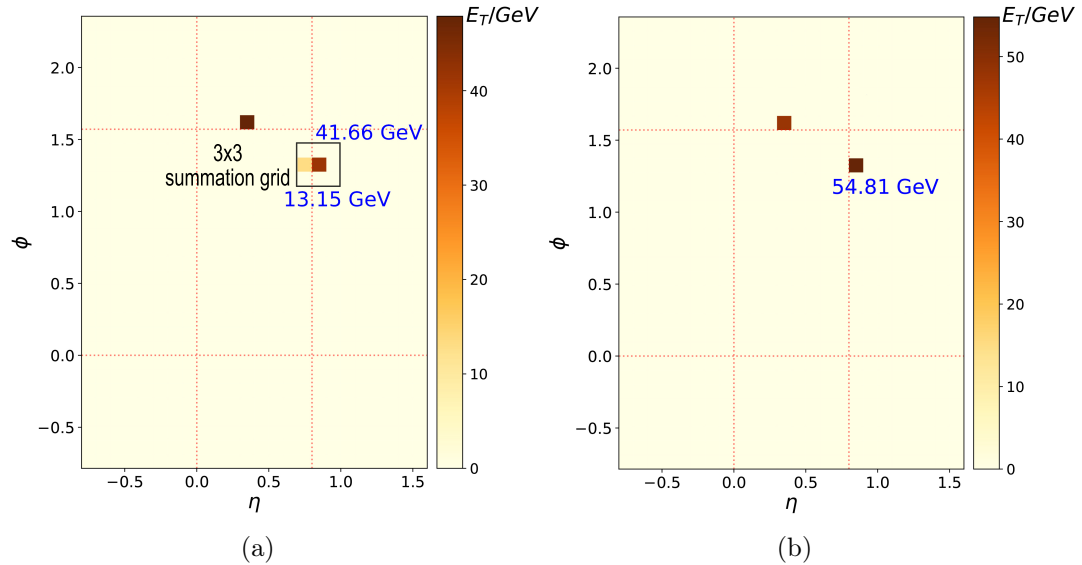


Figure 6.24: If the network fills additional towers within a 3×3 grid around a local maximum, the maximum is replaced by the sum of all towers within the current grid. Setting the eight remaining towers to zero results in a single tower containing the entire jet energy.

The network of Fig. 6.17 will be extended with one additional stage after the output layer, a layer performing the summation operation described above. This additional layer will be disabled during training, but invoked for evaluation purposes. In that way, the network's output will always have the desired format and will be used in that mode for all following analyses.

Performance Analysis on $t\bar{t}$ Fast Simulation Data

Before analyzing the network's performance on full ATLAS Monte Carlo data, the performance will first be evaluated on the $t\bar{t}$ fast simulation evaluation dataset without pile-up to compare the regression-grid approach with the previous output-list approach. Unless otherwise stated all following analyses are based on close-by jet selections since these cases will be the most challenging for the algorithms.

Fig. 6.25 shows the angular resolutions in η and ϕ with respect to anti- k_t truth for the neural network and anti- k_t online which are now resolved in 0.1 steps as prescribed by the output grid granularity. For both quantities, the neural network peaks higher than anti- k_t online and is hence more accurate in predicting the angular position. The network is not only able to reach anti- k_t online angular precision but is able to outperform the algorithm in that respect.

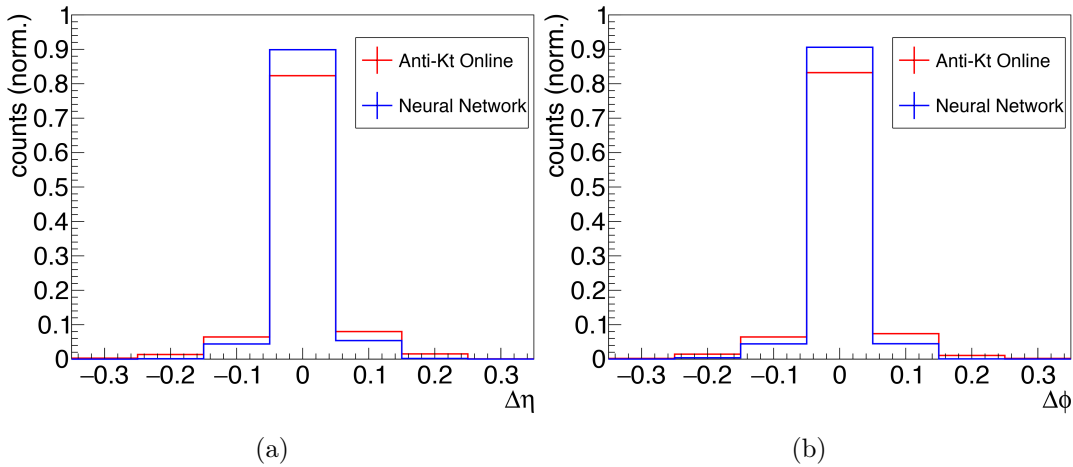


Figure 6.25: Angular resolution in η and ϕ for the neural network and anti- k_t online, defined as the difference to the anti- k_t truth values:

$$\Delta\eta = (\eta_{\text{truth}} - \eta) \text{ and } \Delta\phi = (\phi_{\text{truth}} - \phi).$$

The energy resolutions of both algorithms with respect to the true values are shown in Fig. 6.26. As one can see, anti- k_t online performs slightly better in resolving the correct jet energies. Due to the training artifact described above, the network is not fully capable of learning to reconstruct the exact energy values. The distribution is asymmetric towards larger values, i.e. the network tends to predict smaller jet energies. The worse resolution in comparison to anti- k_t online mostly results from the fact that the sum of multiple predicted towers does not exactly match the true jet energy. This can be seen in the energy resolution for those neural network jets that

were correctly predicted as one single tower in the output grid, which is also depicted in the figure below. For these jets, the resolution is almost as good as the one of anti- k_t online with less asymmetry compared to all neural network jets.

Even though anti- k_t online peaks high at zero, it also comes with a tail for negative ΔE_T values which results from the algorithm merging close jets and thus predicting too large energy values.

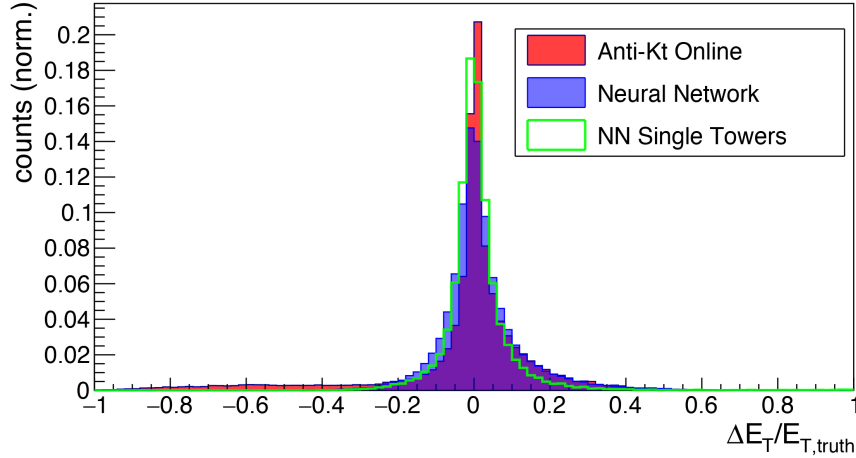


Figure 6.26: Energy resolutions for jets found by antikt online, the neural network and for only correctly predicted neural network jets (NN Single Towers), defined as the difference to the anti- k_t truth values: $\Delta E_T = E_{T,\text{truth}} - E_T$. The jet energies vary between $20 \text{ GeV} < E_{T,\text{truth}} < 2 \text{ TeV}$.

In comparison to the previous output-list approach, the regression-grid network performs considerably better, even if it does not fully reach the anti- k_t online energy resolution. The energy resolution distributions are again fitted using Gaussian fit functions, which are for clarity reasons not depicted in Fig. 6.26, allowing to emphasize the better performance quantitatively. The corresponding fit parameters are listed in Table 6.3.

Algorithm	$p_0/10^{-1}$	$\mu/10^{-3}$	$\sigma/10^{-2}$
anti- k_t online	0.99 ± 0.01	-4.23 ± 0.37	5.86 ± 0.06
neural network	0.89 ± 0.02	3.11 ± 0.27	6.85 ± 0.04
NN Single Towers	0.95 ± 0.02	0.47 ± 0.13	4.57 ± 0.03

Table 6.3: Summary of Gaussian fit parameters p_0 (scaling factor), μ (mean value) and σ (standard deviation) for the energy resolutions in Fig. 6.26.

The better performance in comparison to the previous approach is also reflected in the efficiency curves of both algorithms, as seen in Fig. 6.27.

The network's efficiency is very close to the one of anti- k_t online in the case of all-jet selection, illustrated in Fig. 6.27(b). Both algorithms feature a steep slope and eventually reach 100% efficiency for energies $E_T > 300 \text{ GeV}$. The reason why both

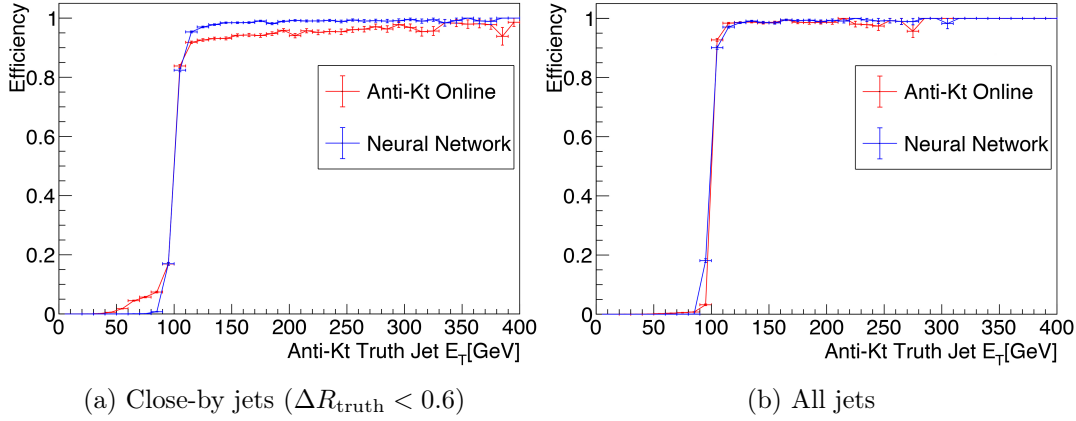


Figure 6.27: Efficiency curves for the neural network and anti- k_t online with respect to anti- k_t truth jets with $E_T > 100$ GeV in the FPGA core region. A jet is only considered found if the algorithm predicts its position within the FPGA core region.

algorithms have difficulties reaching the plateau is due to their angular uncertainty and the selection of only jets in the FPGA core region: Truth jets whose centers are close to the border but still within the FPGA core region are sometimes predicted to be outside of the core region by the neural network and anti- k_t online and therefore not found.¹ Efficiency curves for jets only in the innermost core region ($0.2 < \eta < 0.6$ and $1/8\pi < \phi < 3/8\pi$), where the angular resolution does not affect the plateau, are given in Fig. A.3.

For the challenging cases of overlapping close-by jets (Fig. 6.27(a)), the neural network has a better efficiency than anti- k_t online. Two regions can be identified where the network performs better: $E_T < 90$ GeV and $E_T > 110$ GeV. This arises mostly due to anti- k_t online merging two jets that are relatively close to each other ($\Delta R \approx 0.4$). In these cases, the true jets are separated far enough in order not to be merged by anti- k_t truth ($\Delta R_{\text{truth}} > 0.4$) but due to the discrete input information with coarse granularity, anti- k_t online reconstructs them as only one single jet. Although the neural network receives the same coarse granularity inputs as anti- k_t online, it has learned to use these discrete information to draw conclusions on the underlying truth information provided by anti- k_t truth.

Fig. 6.28 illustrates this behavior for one example event in the first of the two regions with $E_T < 90$ GeV.

As established above, the neural network performs well on unseen fast simulation $t\bar{t}$ data and is able to outperform the anti- k_t online algorithm in terms of angular resolution and jet finding efficiency.

The algorithm's performance on ATLAS full simulation Monte Carlo data, however, remains to be investigated. Since the network was trained on fast simulation data, an evaluation on full simulation samples will impose new challenges on the algorithm. In the following, the neural network's performance is analyzed on ATLAS Di-Higgs and

¹In the final application where the algorithms are applied on each FPGA subregion separately, these jets are then however found by the neighboring FPGA and therefore not lost.

$t\bar{t}$ full simulation data.

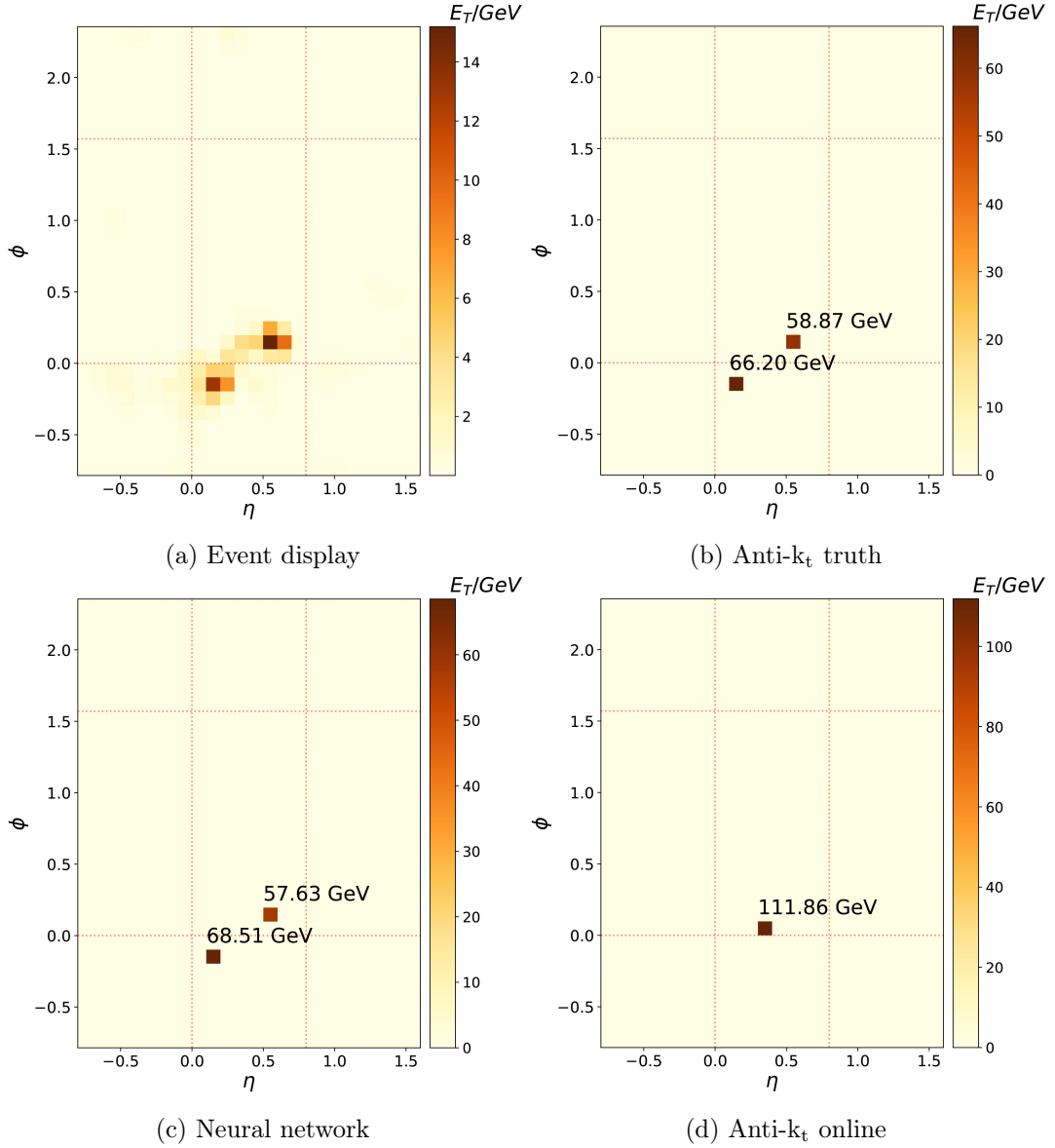


Figure 6.28: Example event of two overlapping anti- k_t truth jets from the $t\bar{t}$ fast simulation dataset. The neural network is able to reconstruct two single jets with good energy and perfect angular agreement in comparison to the truth values, while anti- k_t online reconstructs only one jet with merged energies.

Performance Analysis on ATLAS Di-Higgs Full Simulation Data

The neural network is now applied on ATLAS full simulation Di-Higgs data with pile-up of $\langle\mu\rangle = 60$. Fig. 6.29 shows an example event display from this evaluation dataset. Note that the energy differences between anti- k_t truth and the other both algorithms arise due to a different energy calibration which results in calorimeter jets having roughly half the energy of the jets reconstructed from truth information.

Even though the network has been trained on $t\bar{t}$ fast simulation data without pile-up, it is able to cope with this new type of data and reconstruct all truth jets of the shown event. Again, the network can resolve two close-by jets (left) that anti- k_t online merges into one single jet.

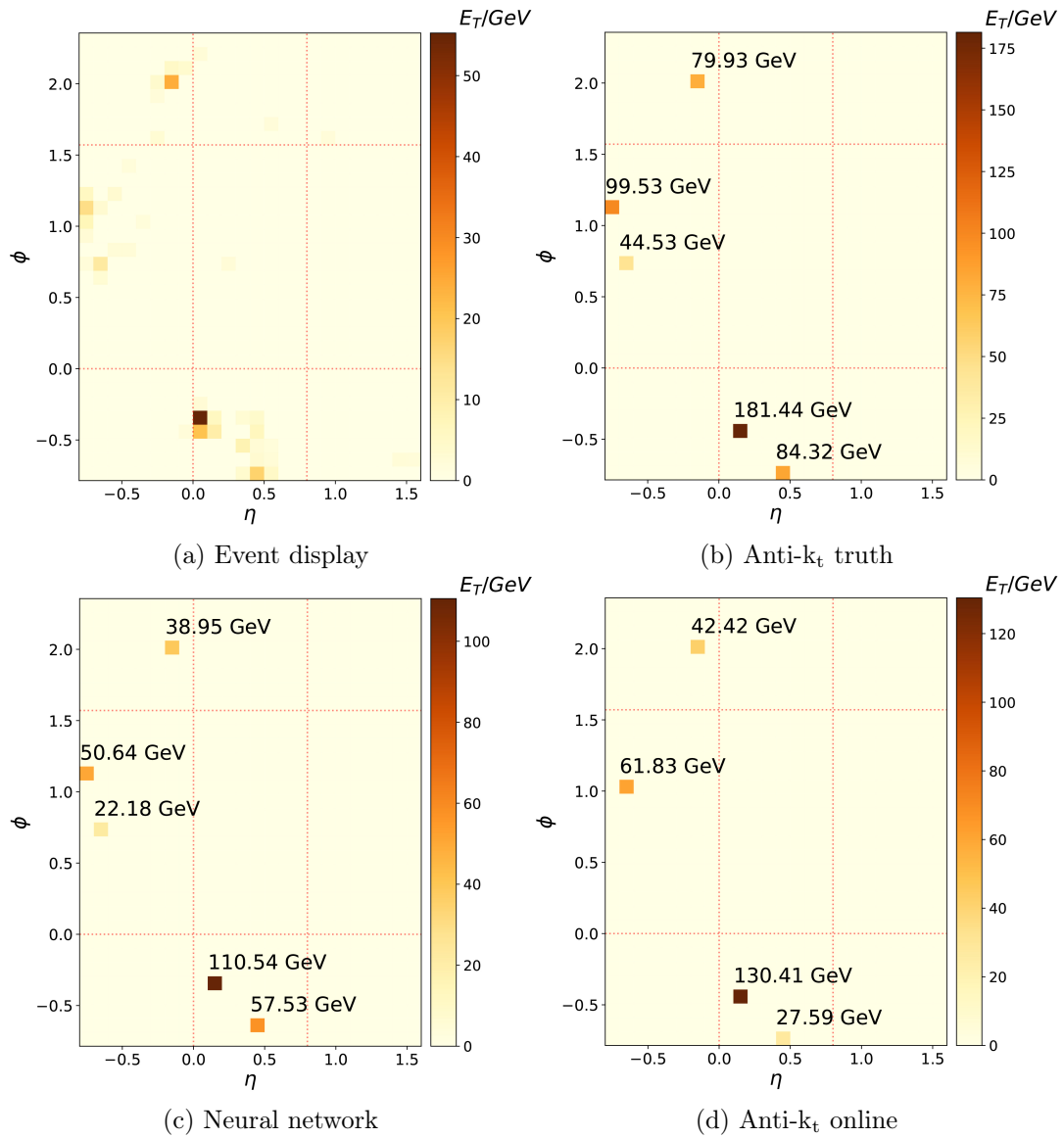


Figure 6.29: Example event from the ATLAS Di-Higgs full simulation dataset. The neural network is able to reconstruct all truth jets, while anti- k_t online merges two jets. The energy differences compared to anti- k_t truth are due to different energy calibrations.

Due to the different energy calibrations, energy resolution distributions are not meaningful anymore and will therefore not be studied. However, η and ϕ angular resolutions are given in Fig. 6.30. The neural network is still slightly more accurate in predicting the jet positions in comparison to anti- k_t online, although the resolutions for both algorithms worsened compared to the fast simulation $t\bar{t}$ case.

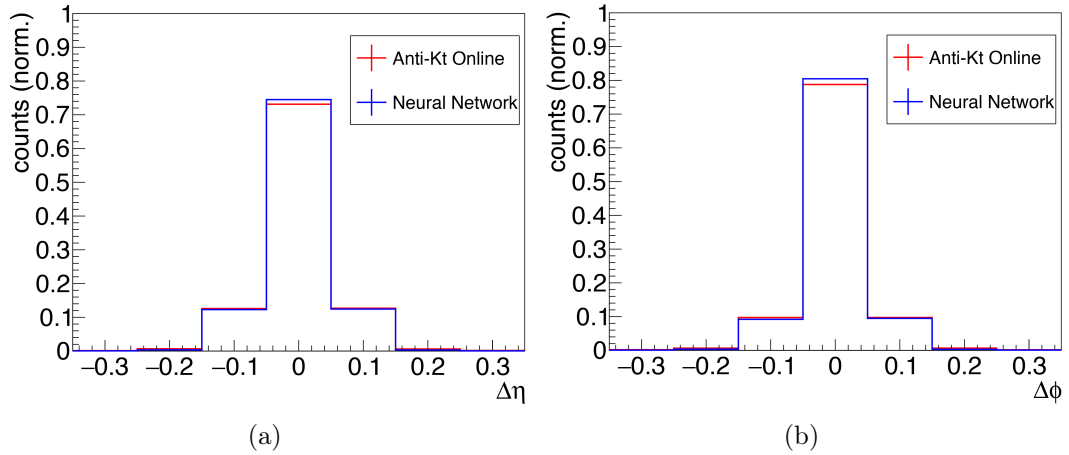


Figure 6.30: Angular resolution in η and ϕ for the neural network and anti- k_t online, defined as the difference to the ATLAS full simulation anti- k_t truth values: $\Delta\eta = (\eta_{\text{truth}} - \eta)$ and $\Delta\phi = (\phi_{\text{truth}} - \phi)$.

Also the efficiency on full simulation data decreases for both algorithms, as seen in Fig. 6.31. The neural network performs better for close-by jets and both algorithms provide almost similar performance for the selection of all jets. In both cases, anti- k_t has a slightly steeper slope, meaning that it still features a better energy resolution than the network.

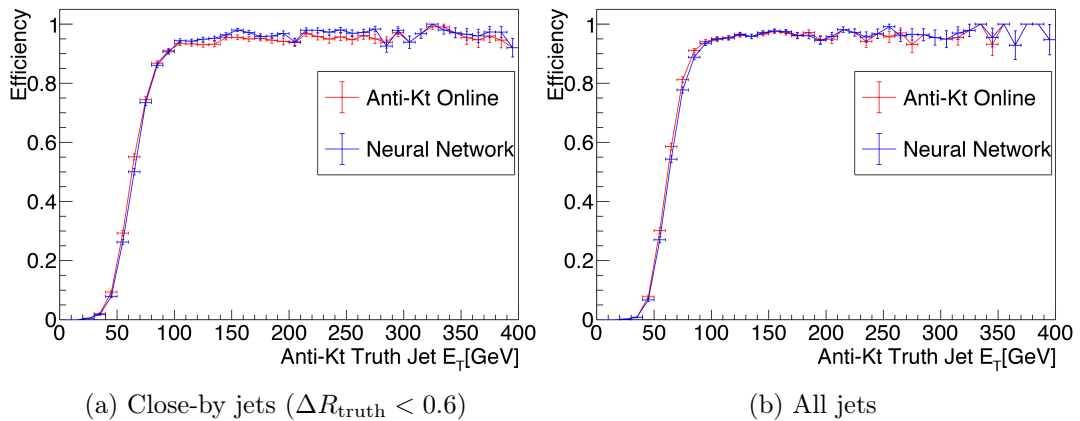


Figure 6.31: Efficiency curves for the neural network and anti- k_t online with respect to anti- k_t truth jets for $E_T > 30$ GeV in the FPGA core region. A jet is only considered found if the algorithm predicts its position within the FPGA core region.

The plateaus are not reached since jets close to the core region border are sometimes predicted to be in the environment and therefore lost due to the finite angular resolutions of both algorithms. These jets are then, however, found in the neighboring

FPGA regions. Hence, it is reasonable to analyze the jet finding performance on the entire calorimeter², meaning that the algorithm is applied on every FPGA subregion separately and the individual results will finally be merged to give one combined calorimeter jet collection. After all, this combined jet collection of the entire event will be the entity of interest for calorimeter trigger efficiency studies.

The following jet finding efficiency curves will therefore be rate-matched in such a way that both algorithms yield the same trigger rates of 8 kHz. The online thresholds are then derived accordingly. Fig. 6.32(a) shows the single-jet trigger rates for the neural network and anti- k_t online, Fig. 6.32(b) the 3-jet trigger rates for both algorithms.

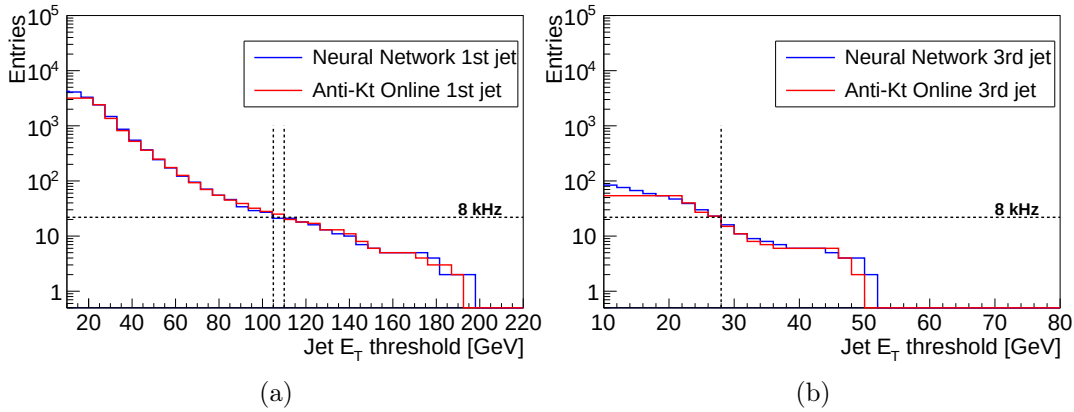


Figure 6.32: (a) Single-jet trigger rates and (b) 3-jet trigger rates for the neural network and anti- k_t online.

Fig. 6.33 and 6.34 show the corresponding rate-matched efficiency curves for the neural network, anti- k_t online as well as for offline reconstructed anti- k_t jets.

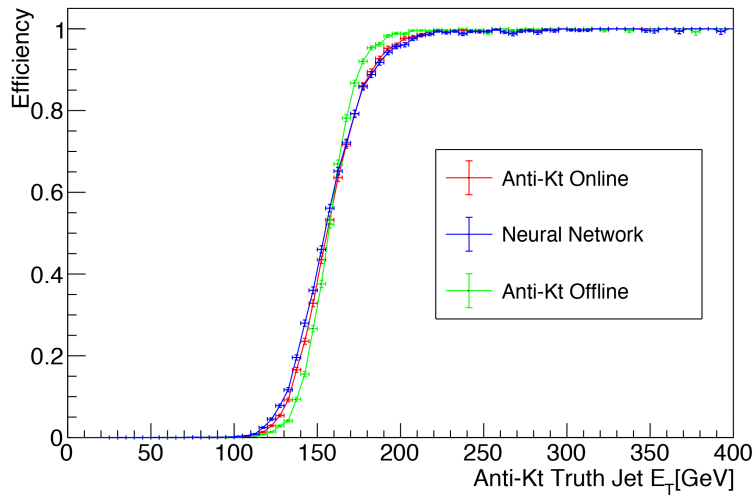


Figure 6.33: Rate-matched single-jet trigger efficiency curves for neural network, anti- k_t online and anti- k_t offline.

For the demanded trigger rates of 8 kHz, the resulting single-jet online thresholds are $E_T^{\text{cut}} = 105 \text{ GeV}$ for the neural network and 110 GeV for anti- k_t online while

²Only in the central region for $|\eta| < 2.4$, i.e. the forward calorimeter is not taken into account.

$E_T^{\text{cut}} = 28 \text{ GeV}$ is used for the multi-jet trigger as the third jet threshold for both algorithms. Since offline and truth jet energies are calibrated differently from online jet energies, the calibration was taken into account by applying an offline jet threshold in such a way that the points with 50% efficiency of all algorithms coincide. The chosen offline jet energy thresholds are 155 GeV for the single-jet trigger and 50 GeV for the multi-jet trigger.

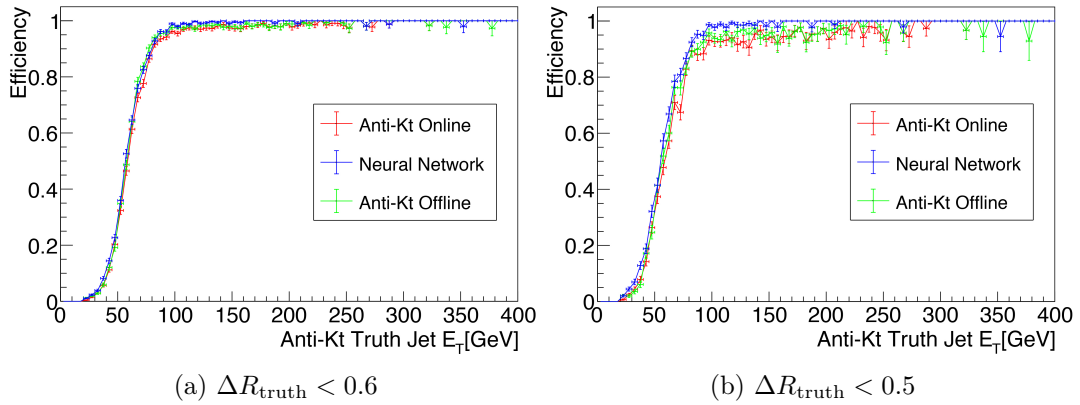


Figure 6.34: Rate-matched 3-jet trigger efficiency curves for the neural network, anti- k_t online and anti- k_t offline for two different close-by selections, $\Delta R_{\text{truth}} < 0.6$ and $\Delta R_{\text{truth}} < 0.5$.

Comparing anti- k_t online and anti- k_t offline with the neural network's performance in finding close-by jets (Fig. 6.34), one can see that the network performs the best. It clearly outperforms anti- k_t online and also reaches a better efficiency than anti- k_t offline, although anti- k_t offline's slope is slightly steeper due to a better energy resolution. Considering the fact that the neural network only receives coarse calorimeter information as input while the offline reconstruction is based on full detector information, the network's performance is remarkably effective for close-by jets.

However, anti- k_t offline performs the best in the case of all-jet selection (Fig. 6.33) whereas the network shows small inefficiencies also for high energies, i.e. it loses a small fraction of jets. The reason for this behavior will be explained in the following.

Fig. 6.35 shows an example event display with one lost neural network Di-Higgs jet close to a boundary between two FPGAs. In fact, the jet is initially found by the neural network but lost in recombining all calorimeter subregions into one common jet collection, as explained in the following. Fig. 6.36 shows the unsummed neural network predictions of the two neighboring FPGA subregions where the original event features the jet that will eventually be lost. In both cases, the network finds the jet but due to its finite energy resolution and the tower energies being close to each other, it places the towers with the maximum energy value in different positions, i.e. both times exactly in the neighboring FPGA core region.

Since the final jet position after the network's summation layer will be the one containing the local maximum, the jets are predicted in the environment in both cases and therefore never placed in a core region. That way, the jet is lost since jets in the

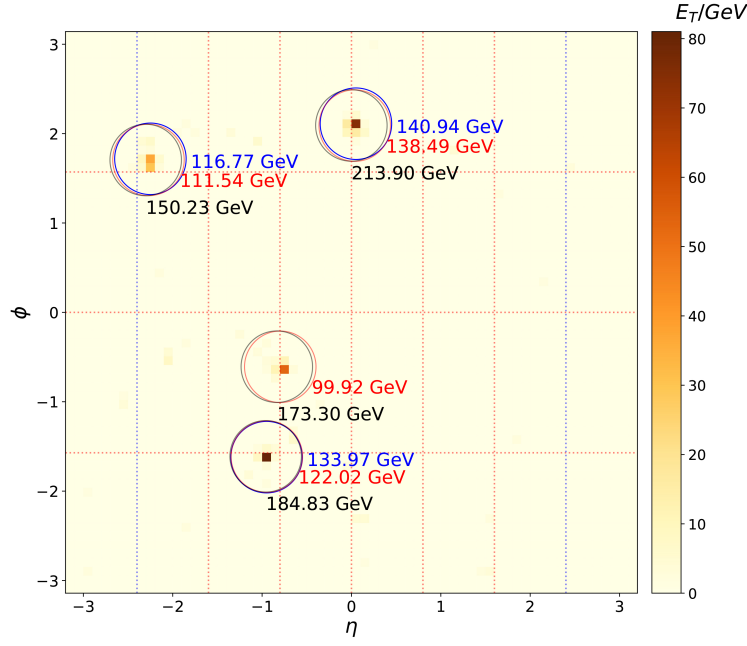


Figure 6.35: Example Di-Higgs event display showing anti- k_t truth (black), anti- k_t online (red) and neural network (blue) jets with their corresponding energies. The neural network jet at $(\eta, \phi) \approx (-0.8, -0.6)$ is lost during the recombination of all FPGA subregion jets into one calorimeter jet collection.

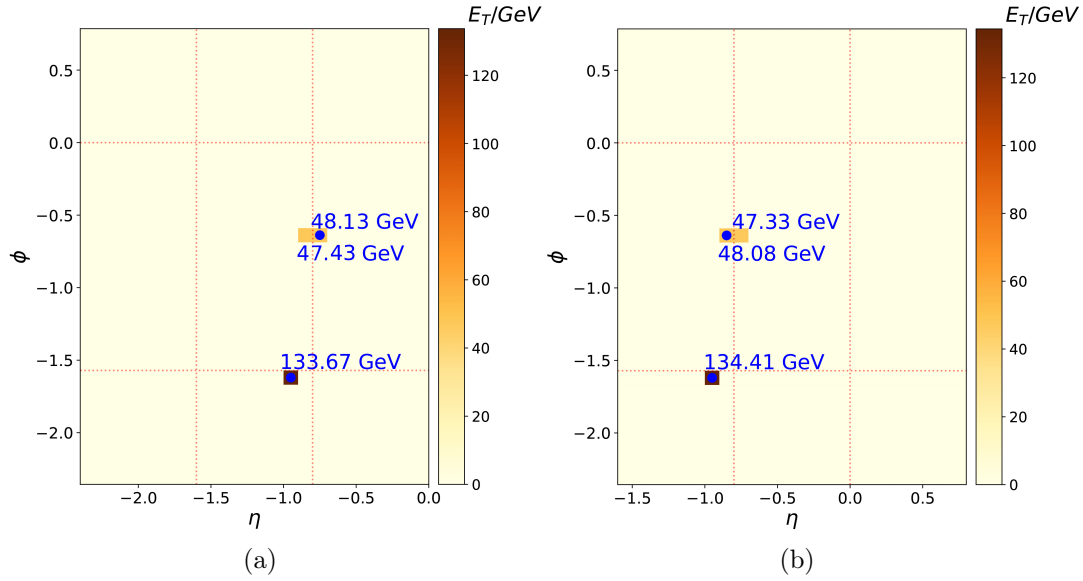


Figure 6.36: Unsummed neural network predictions for a jet close to the border between two FPGA regions. Both times, the tower containing the local maximum and therefore the final jet position (blue dot) is placed in the environment, resulting in an overall lost jet.

environment are supposed to be found by the neighboring FPGA and thus discarded. This behavior occurred 63 times out of about 50 000 Di-Higgs events.

A possible solution could involve extending the core region for finding jets by one tower in each direction and discarding potentially double-counted jets afterwards.

Since the network’s architecture is entirely based on convolutional – and thus translational invariant – operations, it can also be utilized to make predictions on the entire calorimeter, i.e. the input to the exact same network can be chosen to be 48×64 towers in $\Delta\eta \times \Delta\phi$ for evaluation purposes instead of the FPGA subregion size of 24×32 towers.

The efficiency curve in Fig. 6.37 shows the network’s performance when being applied on the entire calorimeter at once. The inefficiencies for high energies identified above are not present anymore, confirming that they indeed arose due to losing jets at the FPGA boundaries.

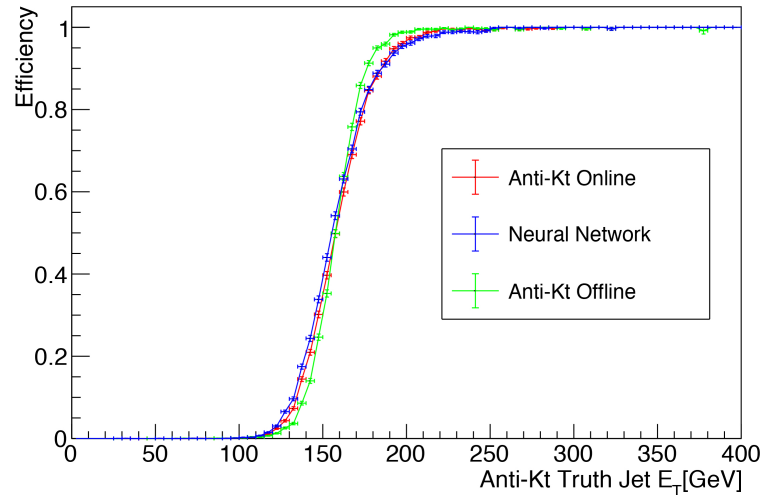


Figure 6.37: Single-jet trigger efficiency curves for the neural network applied on the entire calorimeter at once, anti- k_t online and anti- k_t offline.

Performance Analysis on ATLAS $t\bar{t}$ Full Simulation Data

Fig. 6.38 shows the single-jet trigger efficiency curves for all three algorithms on $t\bar{t}$ full simulation data, where the neural network is again applied on each FPGA subregion separately. Anti- k_t offline still performs the best in case of all-jet selection while the neural network and anti- k_t online offer similar efficiencies.

Again, the network outperforms both anti- k_t offline and online in reconstructing close-by jets as shown in Fig. 6.39. The efficiencies of the anti- k_t algorithms decrease with decreasing jet distances while the network’s performance remains stable also for largely overlapping jets ($\Delta R_{\text{truth}} < 0.5$).

The neural network’s performance is not significantly better than its performance on Di-Higgs data, meaning that – despite being trained exclusively on $t\bar{t}$ data – it performs well, independent on the underlying physics process. Accordingly, it has learned to generally reconstruct jets rather than the underlying physics presented during training which was the desired goal.

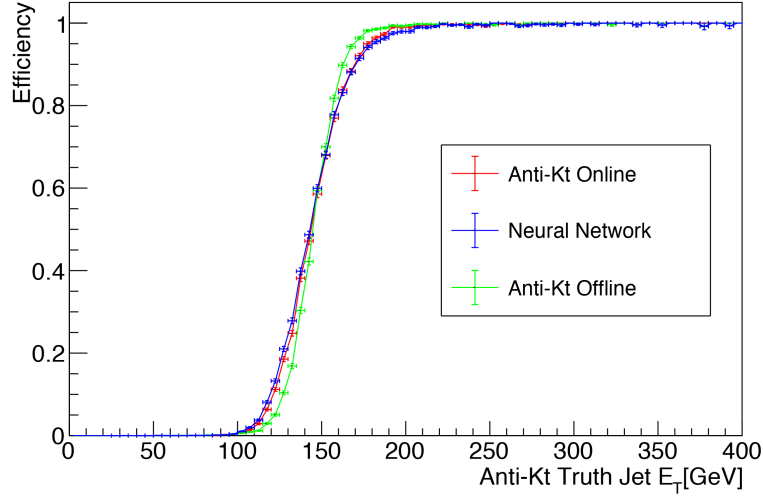


Figure 6.38: Rate-matched single-jet trigger efficiency curves on $t\bar{t}$ full simulation samples for the neural network, anti- k_t online and anti- k_t offline.

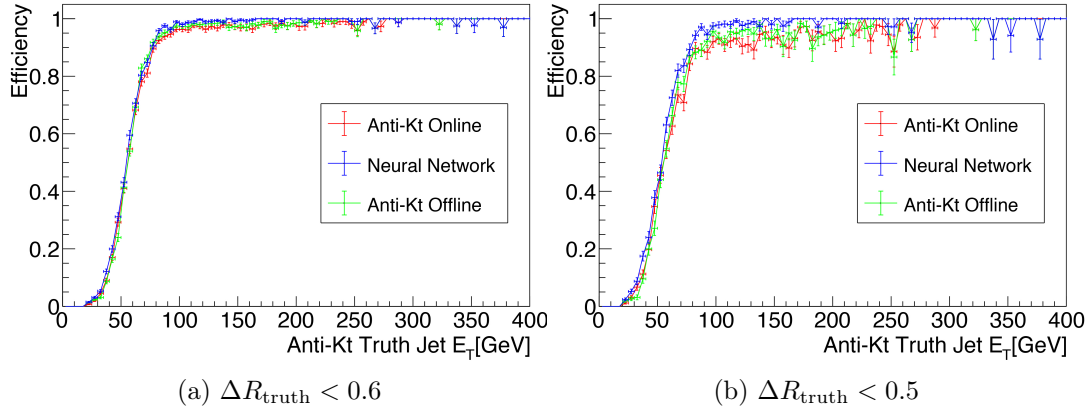


Figure 6.39: Rate-matched 3-jet trigger efficiency curves on $t\bar{t}$ full simulation samples for the neural network, anti- k_t online and anti- k_t offline for two different close-by selections, $\Delta R_{\text{truth}} < 0.6$ and $\Delta R_{\text{truth}} < 0.5$.

6.2.4 Summary of Presented Approach

The regression-grid neural network approach features a different output format compared to the previously presented output-list approach, which helps to diminish training-induced artifacts. Even though also this approach comes with a small training artifact, predicting multiple neighboring pixels instead of one single pixel, it can easily be resolved by appending a final summation layer to the network.

The network performs very well on new data. Especially the fact that the network has been trained on fast simulated $t\bar{t}$ data without pile-up and is able to outperform the state-of-the-art jet clustering anti- k_t online algorithm on full simulated high pile-up Di-Higgs data is remarkable.

The network's biggest weakness compared to anti- k_t is its energy resolution which results from the multiple pixel prediction artifact. It leads to the efficiency curve's

slope being less steep in comparison to anti- k_t and also to small inefficiencies for high energies due to a small fraction of lost jets at the FPGA boundaries.

The neural network has an outstanding performance for reconstructing overlapping jets. Here, it is able to outperform not only anti- k_t online but also the anti- k_t offline algorithm which reconstructs jets using the full available detector information.

Chapter 7

Adaption to Hardware Resources

The regression-grid neural network presented in the previous chapter performs well and would be a good candidate to be used as the ATLAS L1Calo trigger jet finding algorithm implemented on FPGAs. Studies on how these kinds of neural networks can be implemented on FPGAs as well as determinations of their maximum sizes in terms of depth and network parameters are currently being conducted. Rough estimates on the maximum possible amount of arithmetic operations for a network to be implemented on FPGAs are already available for a network architecture as used in Section 6.2.2. These numbers are of course highly dependent on the utilized hardware and will therefore be analyzed separately.

The computationally most expensive arithmetic operations on FPGAs needed for a neural network forward pass are multiplications and are thus the major limiting factor in realizing an implementation on FPGAs. The maximum possible network size will therefore be given in terms of required weight multiplications $w_i \cdot x_i$ for one single forward pass through the entire network.

Three different cases will be considered in the following:

1. Phase-1 hardware with Xilinx UltraScale+ FPGAs (XCVU9P). The maximum number of weight multiplications is estimated to be between 100 000 (using Float16 parameters, high precision) and 450 000 (fixed-point, low precision).
2. Phase-2 (starting around 2025) hardware with current baseline Xilinx UltraScale+ FPGAs (XCVU13P), between 120 000 (high precision) and 650 000 (low precision) possible weight multiplications.
3. Phase-2 hardware including an *adaptive compute acceleration platform* (ACAP) [31]. Unfortunately, not much information is available yet for this new technology. The maximum number of weight multiplications is roughly estimated to be five times higher compared to FPGA-based Phase-2 hardware. The following study for this case will be based on a best-case scenario assumption with about 3 000 000 possible weight multiplications.

The number of required multiplications can be derived from the network's architecture and its amount of parameters. Especially convolutional layers hide a lot of these

operations for a relatively small amount of parameters: A kernel with size $n \times n$ applied at a single position of an image with N_c channels involves $n^2 \times N_c$ multiplications. This number is multiplied by the amount of possible kernel positions N_{pos} while the kernel is slid across its input image. Finally, a convolutional layer often comprises more than one single kernel. Therefore, the number of multiplications obtained above needs to be multiplied by the amount of kernels N_{ker} used in this layer. The total amount of multiplications for one single convolutional layer is therefore given by

$$N_{\text{mul}} = n^2 \cdot N_c \cdot N_{\text{pos}} \cdot N_{\text{ker}}. \quad (7.1)$$

If padding is used, multiplications where the kernel is applied on the zero padded areas can be omitted.

One can now use these kinds of considerations to determine how many weight multiplications are needed for one single forward pass of the regression-grid network from Section 6.2.2. The total number is calculated to be 17 104 512.

It exceeds the maximum possible values stated above by orders of magnitude. An attempt to drastically reduced the network's size without sacrificing too much of its performance will be presented in the following.

7.1 General Parameter Reduction

First of all, the network's architecture will be reduced to a minimum in the sense that only one down-sampling and one up-sampling step is performed, leading to a smaller architecture of the general form illustrated in Fig. 7.1. Instead of a learnable up-sampling in the form of transpose convolution, ordinary 2×2 up-sampling (unpooling) is used as it does not comprise any multiplications and is computationally very inexpensive.

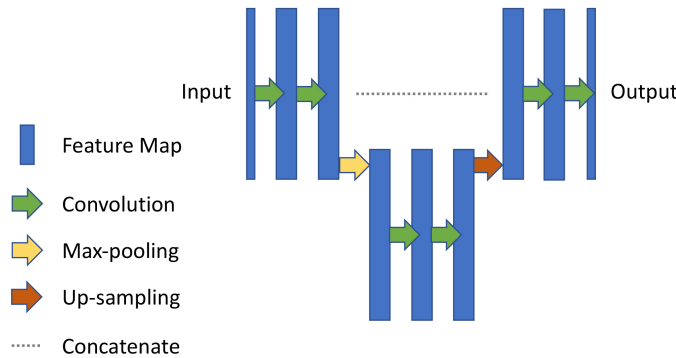


Figure 7.1: Reduced regression-grid architecture for adaption to available FPGA resources.

The exact network's architecture and its hyperparameters, such as kernel sizes and amount of kernels, will be adapted to the respective case individually. For hyperparameter specification, the first two convolutional layer will be denoted with "Conv-1-1"

and “Conv-1-2”, the down-sampled layers with “Conv-2-1” and “Conv-2-2” and the last two convolutional layers with “Conv-3-1” and “Conv-3-2”.

Additionally, L^1 -parameter regularization will be employed, pushing less important parameters close to zero. These parameters can then be discarded for a potential hardware implementation which further reduces the amount of necessary operations.

7.2 Phase-1 Hardware with Xilinx UltraScale+ FPGAs (XCVU9P)

In order to cope with the extremely small amount of possible weight multiplications for L1Calo Phase-1 hardware (Xilinx UltraScale+ XCVU9P FPGA), the general architecture shown above is further reduce by omitting layers *Conv-2-2* and *Conv-3-2*. Hence, the network comprises two convolutional layers following the input layer, one convolutional layer in the down-sampling path and finally one convolutional layer before the output.

The chosen hyperparameters for the network’s architecture are given in Table 7.1, resulting in 921 parameters and 353 456 weight multiplications.

Layer	Amount of Kernels	Kernel Size
Conv-1-1	4	3×3
Conv-1-2	4	3×3
Conv-2-1	4	5×5
Conv-2-2	-	-
Conv-3-1	4	3×3
Conv-3-2	-	-

Table 7.1: Hyperparameters for reduced regression-grid architecture (Fig. 7.1) for adaption to Xilinx UltraScale+ FPGA (XCVU9P).

Since only one down-sampling operation is employed and the network should still be able to observe full jet shapes, the kernels in the down-sampled convolutional layer (Conv-2-1) are enlarged to the size 5×5 .

The network’s parameter distribution is shown in Fig. 7.2. Even though L^1 -parameter regularization was utilized during training, only a few parameters are extremely close to zero, indicating that the network needs its full capacity to solve the given problem and cannot afford discarding many of its weights. Out of all 921 parameters, 4 are smaller than 10^{-3} which is the defined threshold for parameters being small enough to be omitted. According to Eq. (7.1) the required number of weight multiplications scales linearly with the number of parameters in a convolutional layer ($n^2 \cdot N_c$). It is therefore reasonable to assume that $4/921 = 0.43\%$ less weight multiplications and

therefore only

$$353\,456 \cdot (1 - 0.43\%) = 353\,456 - 1\,535 = 351\,921 \quad (7.2)$$

multiplications are needed in total. This number is within the limitations given above and should therefore generally allow an FPGA implementation in terms of network size.

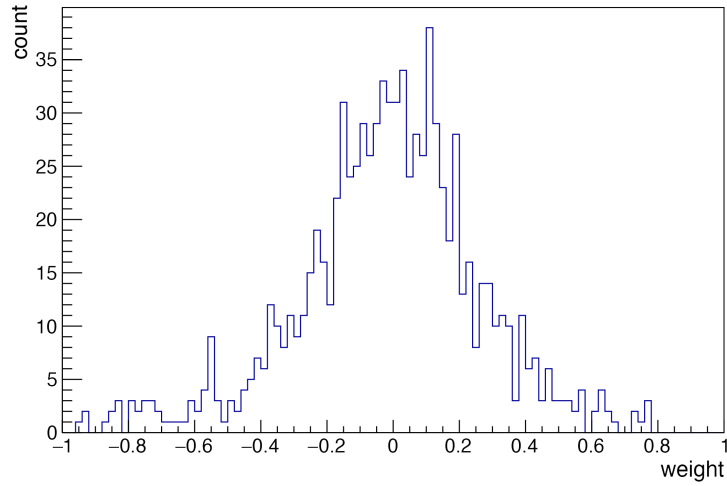


Figure 7.2: Parameter distribution of reduced regression-grid network, adapted to Xilinx UltraScale+ FPGA (XCVU9P) resources.

The rate-matched single-jet trigger efficiency curve for this architecture in comparison with anti- k_t online and anti- k_t offline is shown in Fig. 7.3. The underlying trigger rates are depicted in A.4.

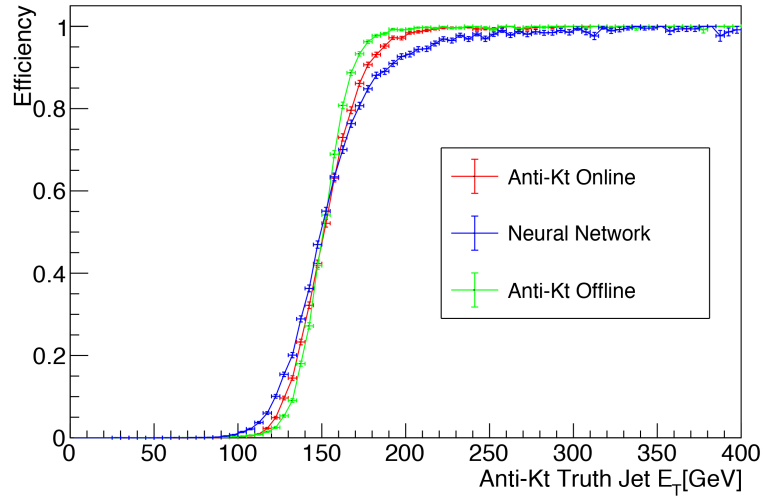


Figure 7.3: Rate-matched single-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and the reduced regression-grid network, adapted to Xilinx UltraScale+ FPGA (XCVU9P) resources.

Since the amount of network parameters – and therefore the network’s capability to properly solve a given problem – was reduced by a factor of about 130 in comparison

to the model presented in the previous chapter, also its performance decreased significantly. The reduced architecture is not large enough for allowing an accurate learning of the demanded task. Especially the worsened energy resolution, reflected in the less steep slope of the efficiency curve, results in inefficiencies for high jet energies due to losing jets at the FPGA boundaries as explained in Section 6.2.3.

The network's performance also decreased for reconstruction of close-by jets, as seen in Fig. 7.4 which shows the efficiency curves for the 3-jet trigger for two different close-by jet selections. Compared to the full regression-grid network, that was able to clearly outperform both anti- k_t algorithms here, the reduced architecture performs worse with efficiencies comparable or even slightly below anti- k_t online.

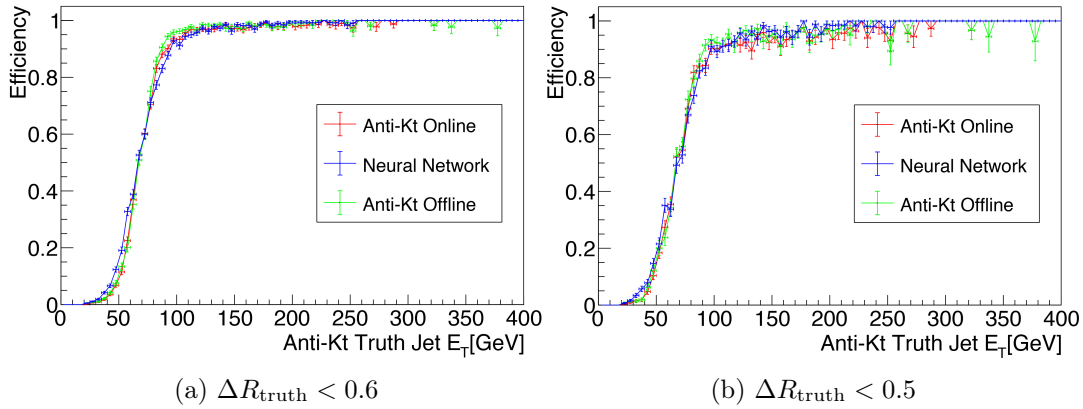


Figure 7.4: Rate-matched 3-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and for the reduced regression-grid network, adapted to Xilinx UltraScale+ FPGA (XCVU9P) resources. The corresponding trigger rates are given in A.4.

7.3 Phase-2 Hardware with Xilinx UltraScale+ FPGAs (XCVU13P)

With the better Phase-2 hardware allowing more weight multiplications, the network's capacity can be slightly increased. Table 7.2 specifies the chosen hyperparameters for adapting the neural network to this particular case.

Layer	Amount of Kernels	Kernel Size
Conv-1-1	4	3×3
Conv-1-2	4	3×3
Conv-2-1	8	5×5
Conv-2-2	-	-
Conv-3-1	8	3×3
Conv-3-2	-	-

Table 7.2: Hyperparameters for reduced regression-grid architecture (Fig. 7.1) for adaption to Xilinx UltraScale+ FPGA (XCVU13P).

Again, layers *Conv-2-2* and *Conv-3-2* are omitted here, but more kernels are added to network. This architecture with the specified hyperparameters results in 1 941 network parameters and 575 312 weight multiplications. The corresponding parameter distribution is shown in A.5. Since 6/1 941 weights are smaller than 10^{-3} , the total amount of required multiplications is determined to be 573 534.

Fig. 7.5 shows the rate-matched single-jet trigger efficiency curve for this architecture. The underlying trigger rates are depicted in A.6.

Since this architecture is still a substantially reduced form of the full regression-grid network and comprises only eight kernels more in total than the network of the previous section, its capacity is still too small and thus it is not able to perform significantly better than the network adapted to Phase-1 hardware.

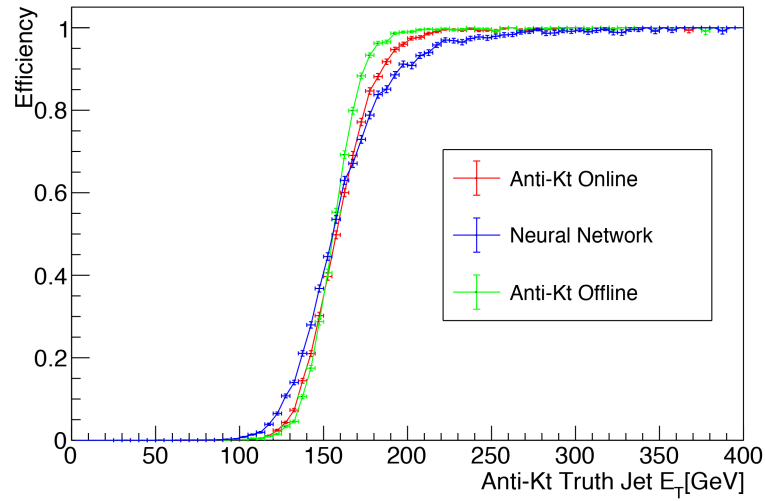


Figure 7.5: Rate-matched single-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and the reduced regression-grid network, adapted to Xilinx UltraScale+ FPGA (XCVU13P) resources.

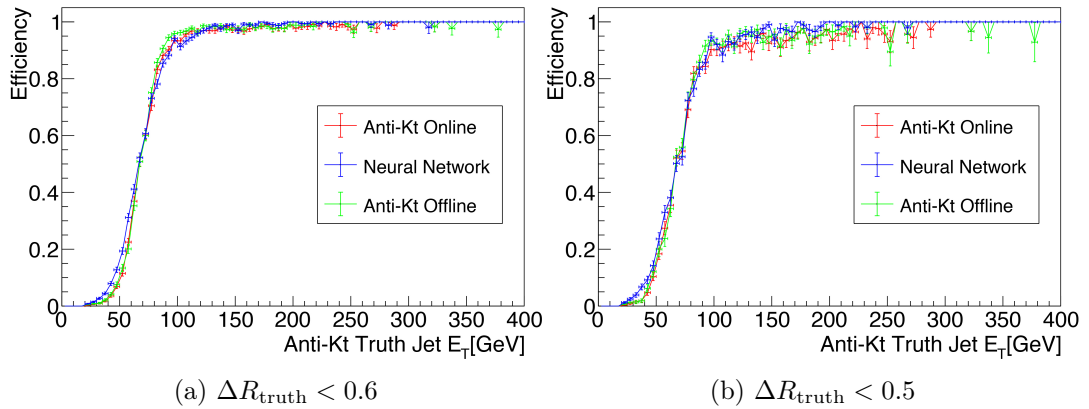


Figure 7.6: Rate-matched 3-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and for the reduced regression-grid network, adapted to Xilinx UltraScale+ FPGA (XCVU13P) resources. The corresponding trigger rates are given in A.6.

7.4 Phase-2 Hardware with ACAP

A potential utilization of adaptive compute acceleration platforms (ACAPs) could significantly increase the amount of possible weight multiplications and therefore the network's capacity. The entire reduced regression-grid architecture as depicted in Fig. 7.1 together with the hyperparameters given in Table 7.4 result in a network with 12 689 trainable parameters and 2 955 744 required multiplications.

The corresponding parameter distribution is shown in A.7. 0.4% of the network's weights are negligibly small, resulting in 2 942 467 necessary weight multiplications for a single network forward pass.

Layer	Amount of Kernels	Kernel Size
Conv-1-1	8	3×3
Conv-1-2	8	3×3
Conv-2-1	16	5×5
Conv-2-2	16	5×5
Conv-3-1	8	3×3
Conv-3-2	8	3×3

Table 7.3: Hyperparameters for reduced regression-grid architecture (Fig. 7.1) for adaption to Phase-2 Hardware with ACAP.

As can be seen in the single-jet trigger efficiency curves in Fig. 7.9, the network performs much better than the two previously analyzed reduced architectures.

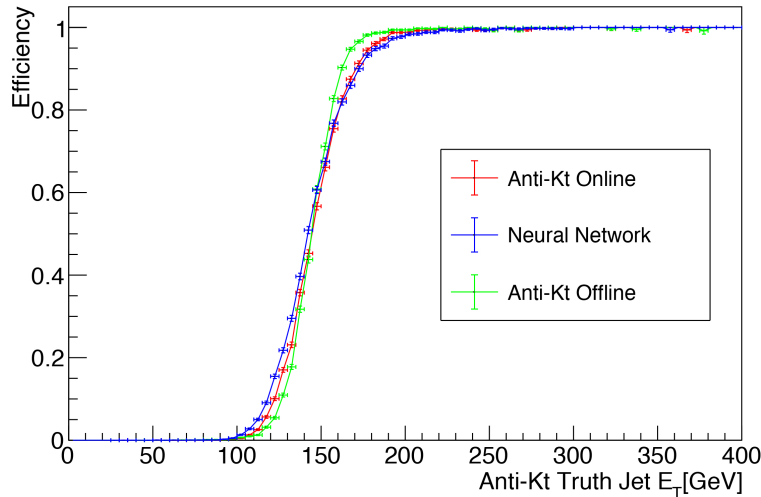


Figure 7.7: Rate-matched single-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and the reduced regression-grid network, adapted to ACAP resources. The underlying trigger rates are given in A.8.

It is nearly as efficient as the full regression-grid network presented in the previous chapter, offering an almost comparable performance to anti- k_t online with a slightly

worse energy resolution.

Similarly, the 3-jet trigger efficiency curves depicted in Fig. 7.8 show that the reduced neural network is able to outperform anti- k_t online for close-by jet reconstruction with $\Delta R_{\text{truth}} < 0.6$ and even outperform anti- k_t offline for $\Delta R_{\text{truth}} < 0.5$. The underlying trigger rates are given in A.8.

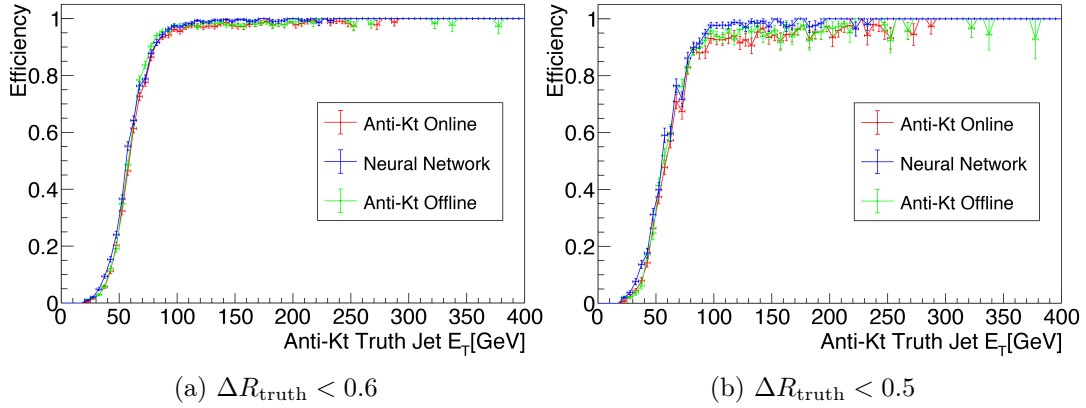


Figure 7.8: Rate-matched 3-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and for the reduced regression-grid network, adapted to ACAP resources. Two different close-by selections, $\Delta R_{\text{truth}} < 0.6$ and $\Delta R_{\text{truth}} < 0.5$, are shown.

Global Calorimeter Input

The Phase-2 upgrade will allow the possibility of receiving global calorimeter information in the trigger. Instead of passing calorimeter subregion information of size 24×32 to the network, the input could then be chosen to be the entire calorimeter, e.g. in the forward region with 48×64 towers in $\eta \times \phi$, eliminating possible subregion boundary losses as explained in the previous chapter. Even though finer granularities will be available in Phase-2, trigger towers of size 0.1×0.1 in $\Delta\eta \times \Delta\phi$ are again assumed for simplicity and reduction of required multiplications. A larger input size comes along with more weight multiplications according to Eq. 7.1 since the number of possible kernel positions increases.

Accordingly, the network's architecture has to be reduced again to fulfill the hardware requirements. Table 7.4 specifies the hyperparameters for an architecture with 2 249 trainable parameters and 2 672 160 multiplications. Only 9 trained parameters are smaller than 10^{-3} , leading to a total amount of 2 661 467 required weight multiplications. The corresponding parameter distribution is shown in A.9.

As opposed to all other neural networks presented in this thesis, this architecture is trained on 48×64 input grids, i.e. it observes the entire calorimeter in the central region at once during training.

Although the network's amount of parameters, and therefore its capability to solve a certain problem, is only slightly larger compared to the network adapted to Phase-2 FPGA resources in Section 7.3, it performs considerably better than this model as can be seen in the efficiency curves in Fig. 7.9 and Fig. 7.10.

Layer	Amount of Kernels	Kernel Size
Conv-1-1	4	3×3
Conv-1-2	4	3×3
Conv-2-1	6	5×5
Conv-2-2	6	5×5
Conv-3-1	4	3×3
Conv-3-2	4	3×3

Table 7.4: Hyperparameters for reduced regression-grid architecture (Fig. 7.1) for adaption to Phase-2 Hardware with ACAP.

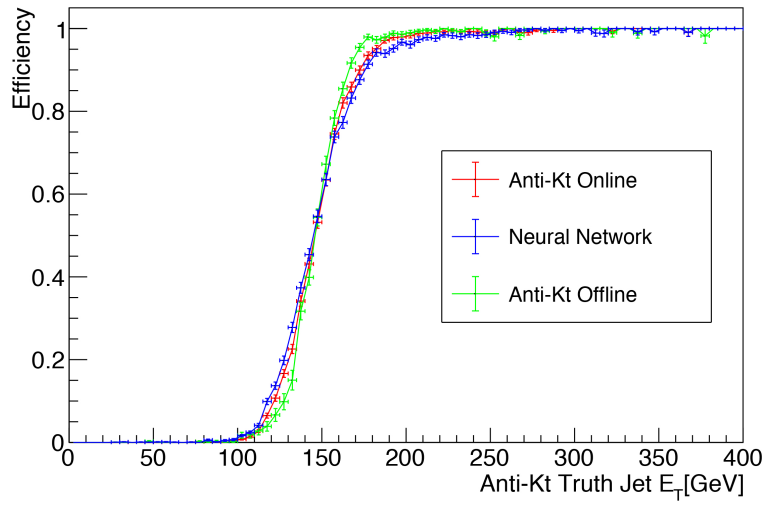


Figure 7.9: Rate-matched single-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and the reduced regression-grid network, adapted to ACAP resources with global calorimeter input. The corresponding trigger rates are given in A.10.

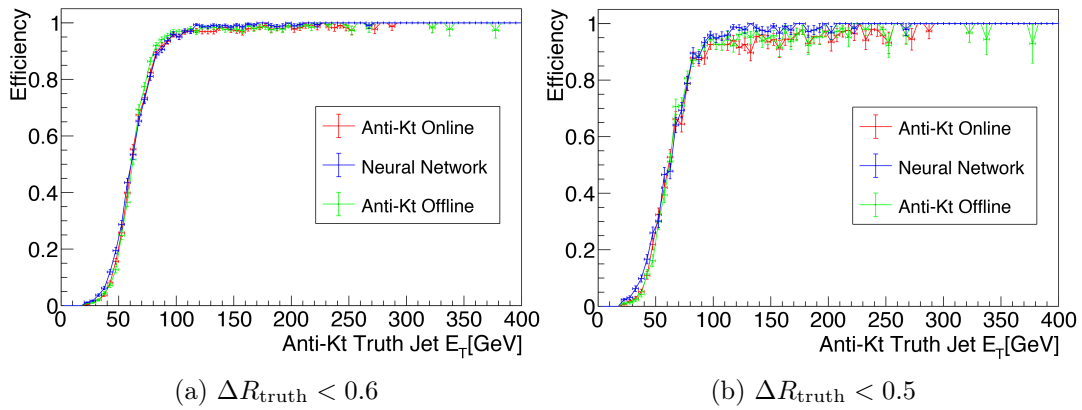


Figure 7.10: Rate-matched 3-jet trigger efficiency curves for anti- k_t online, anti- k_t offline and for the reduced regression-grid network, adapted to ACAP resources with global calorimeter input. Two different close-by selections, $\Delta R_{\text{truth}} < 0.6$ and $\Delta R_{\text{truth}} < 0.5$, are shown.

The corresponding trigger rates are given in A.10.

The slightly larger capacity compared to the Phase-2 FPGA-adapted network helps to increase the energy resolution while the global calorimeter input prevents the algorithm from losing jets at subregion boundaries.

However, the model does not perform better than the ACAP-adapted network with calorimeter subregion inputs. The advantage of having a global calorimeter input is overall not beneficial for the network’s performance as it involves a necessary reduction of its capacity compared to the first ACAP-adapted architecture and thus worsens its ability to perform as efficiently.

The best performing model adapted to the available hardware resources is therefore the network architecture depicted in Fig. 7.1 together with the hyperparameters specified in Table 7.4, i.e. the network adapted to ACAP resources with calorimeter subregion inputs. Its excellent performance makes it a good candidate for L1Calo usage in Phase-2 if the available hardware indeed allows an implementation.

Chapter 8

Summary and Outlook

This thesis presents two different approaches for jet reconstruction with deep neural networks. Both approaches are based on convolutional neural network architectures that differ in their output format. While the network of the first approach outputs an energy sorted list of reconstructed jets, the output of the second approach network is an image itself, containing all relevant jet information.

Although the output-list network is able to find jets for a given calorimeter input region, it comes along with several training-induced characteristics making it not suitable for an eventual utilization.

The second approach's architecture is specifically designed to avoid these kinds of artifacts and hence performs much better. Even though the network has been trained on fast simulation $t\bar{t}$ data without pile-up, it generalizes well and is able to perform very efficiently on ATLAS full-simulated high pile-up Di-Higgs data. The network's greatest strength is the very challenging reconstruction of overlapping jets. Here, it is able to outperform not only the anti- k_t online algorithm but also the offline jet reconstruction algorithm, anti- k_t offline. Considering that the neural network only receives coarse calorimeter information as input while the offline reconstruction is based on full available detector information, the network's performance is remarkably effective for close-by jets. The network's overall performance on the entire evaluation dataset is comparable to the one of anti- k_t online, making it a good candidate for L1Calo usage.

An adaption to very limited hardware resources, however, is not trivial and involves a substantial reduction of the network's capacity. The learnabilities of the FPGA-adapted models for Phase-1 and Phase-2 clearly suffer from the highly reduced network sizes, resulting in lower efficiencies compared to the original model. A utilization of adaptive compute acceleration platforms (ACAPs) in Phase-2 could offer great potential as the ACAP-adapted neural network performs very efficiently, almost comparable to non-reduced regression-grid network.

The use of fast simulation data was sufficient for a proper network training. The excellent reconstruction results on full simulation data for the regression-grid network confirm the high quality of the fast-simulated data that was specifically developed and produced for this thesis. However, fast simulation is never perfect and hence a

source of error. The performances of all networks, and especially the reduced ones with only a small generalization capability, could certainly be further improved by using ATLAS full simulation data for training.

Nevertheless, the work presented in this thesis shows the successful development of a deep neural network architecture that does not only offer a performance similar to the one of anti- k_t online but is even able to outperform this state-of-the-art jet clustering algorithm under certain conditions. A slightly reduced version of this network with almost the same reconstruction efficiency could even be implemented in the ATLAS Level-1 Calorimeter Trigger in Phase-2.

Appendix A

Appendix

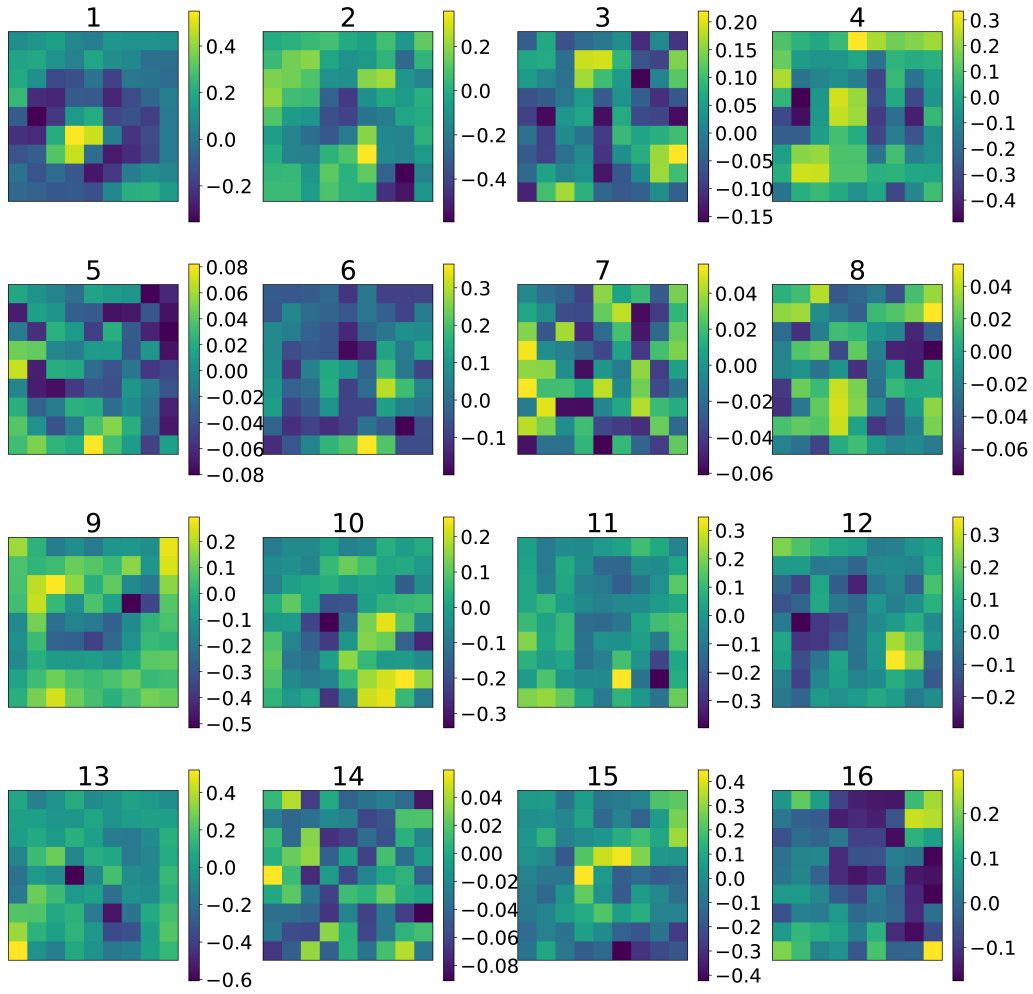


Figure A.1: Illustration of the 16 second layer kernels of single jet finding neural network.

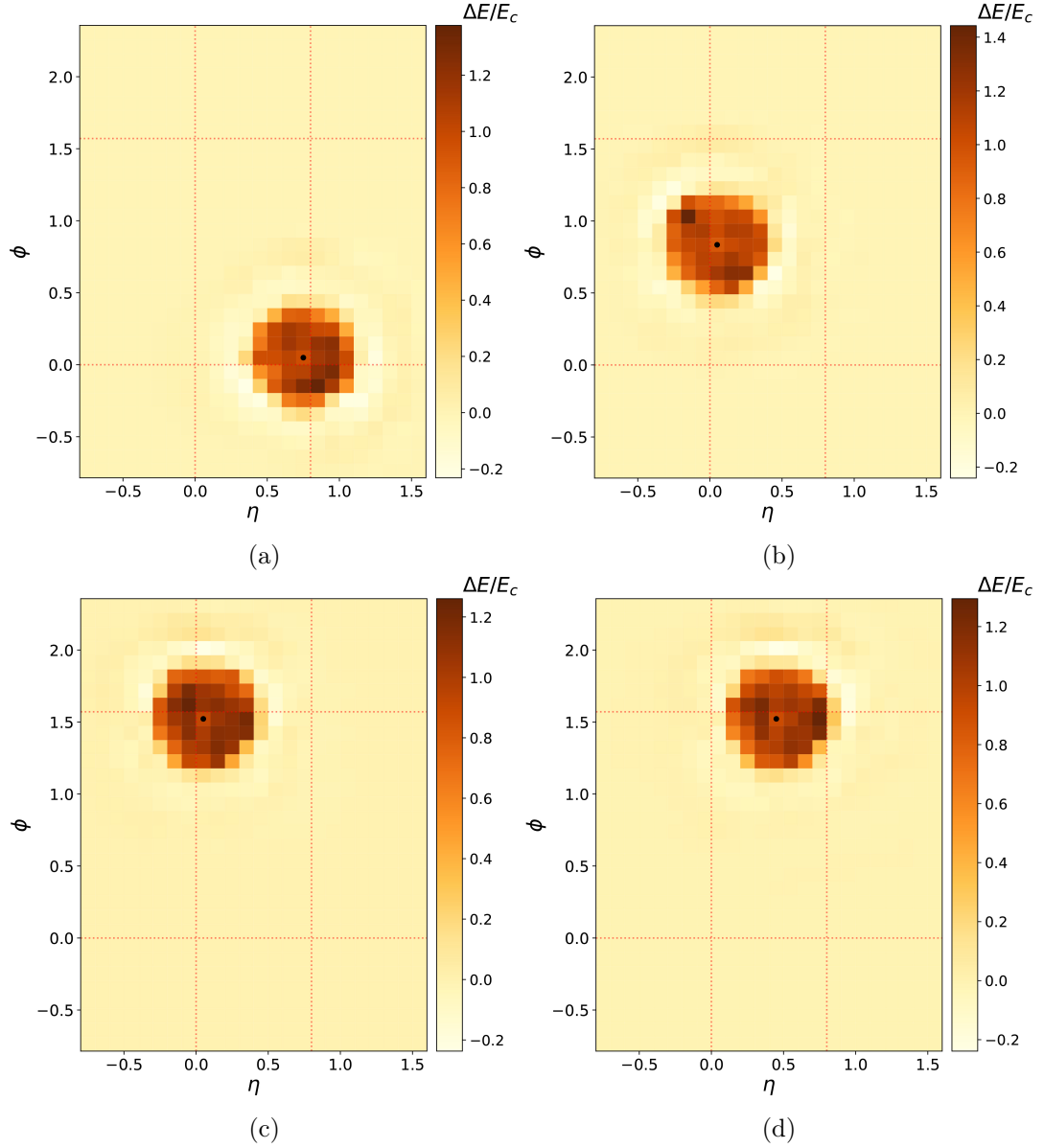


Figure A.2: Influence regions for four different jet positions (black dots), averaged over 50 different jet shapes each. The energy differences ΔE between the original jet energy of 200 GeV and the newly determined once relative to the altered tower energy E_c are shown for the respective tower positions.

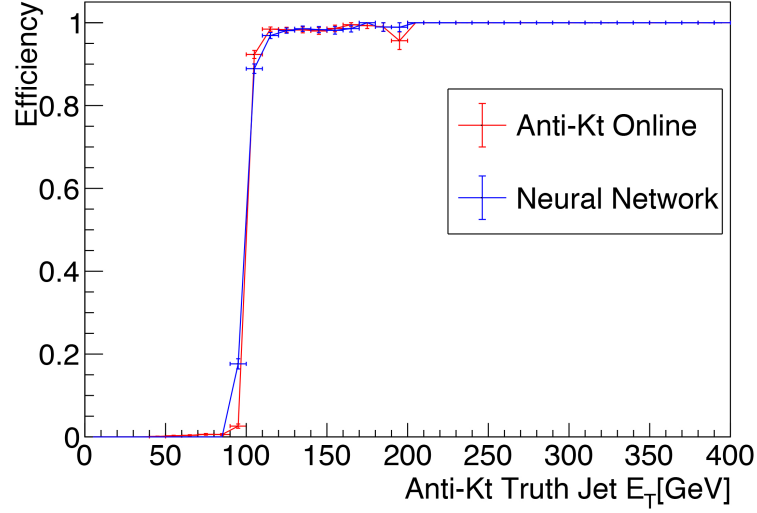


Figure A.3: Efficiency curves for the regression-grid neural network and anti- k_t online with respect to anti- k_t truth jets with $E_T > 100$ GeV the innermost FPGA core region ($0.2 < \eta < 0.6$ and $1/8\pi < \phi < 3/8\pi$) on $t\bar{t}$ fast simulation data.

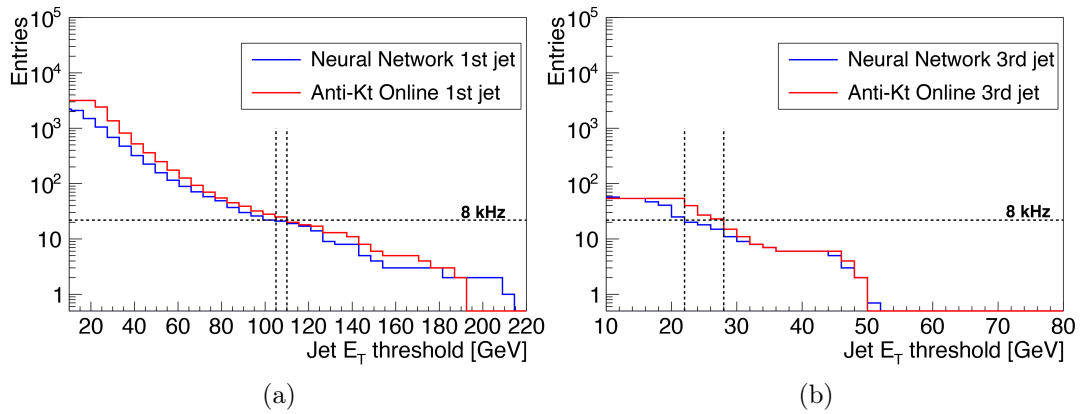


Figure A.4: (a) Single-jet trigger rates and (b) 3-jet trigger rates for the neural network adapted to Phase-1 hardware and anti- k_t online.

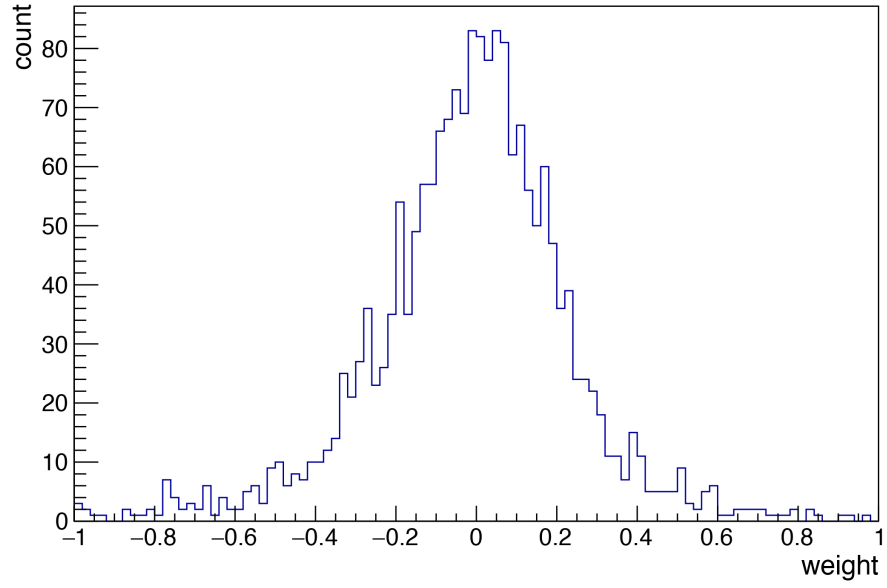


Figure A.5: Parameter distribution of reduced regression-grid network, adapted to Xilinx UltraScale+ FPGA (XCVU13P) resources.

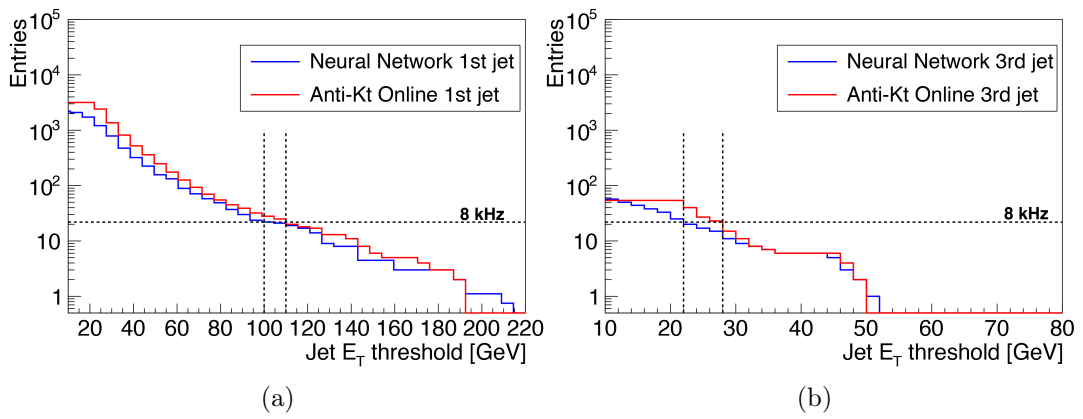


Figure A.6: (a) Single-jet trigger rates and (b) 3-jet trigger rates for the neural network adapted to Phase-2 FPGA resources and anti- k_t online.

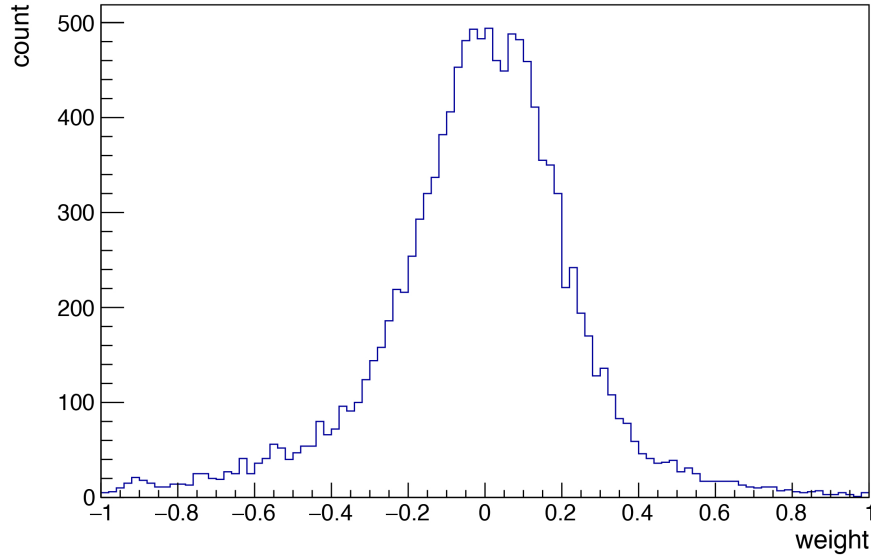


Figure A.7: Parameter distribution of reduced regression-grid network, adapted to Phase-2 ACAP resources.

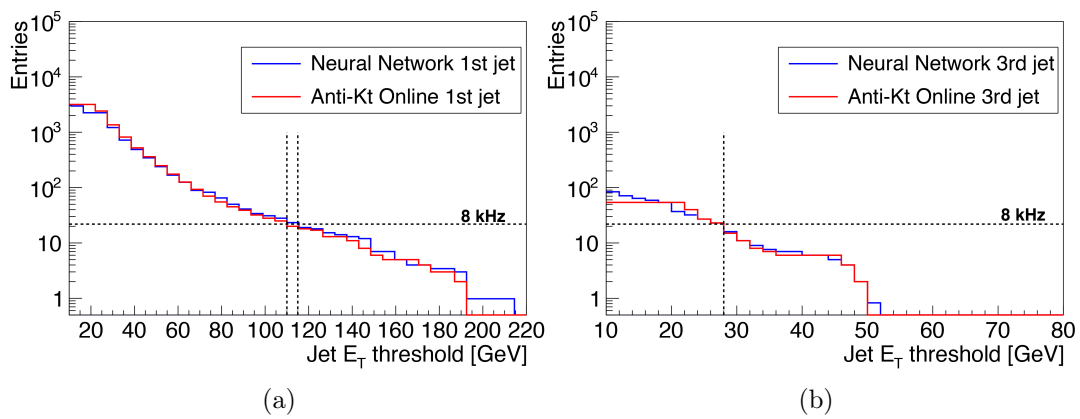


Figure A.8: (a) Single-jet trigger rates and (b) 3-jet trigger rates for the neural network adapted to Phase-2 ACAP resources and anti- k_t online.

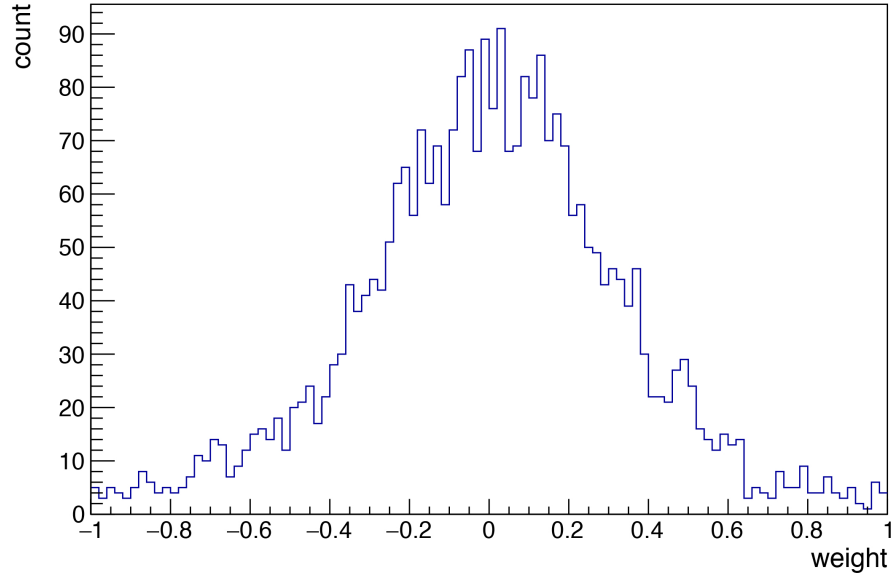


Figure A.9: Parameter distribution of reduced regression-grid network, adapted to Phase-2 global calorimeter ACAP resources.

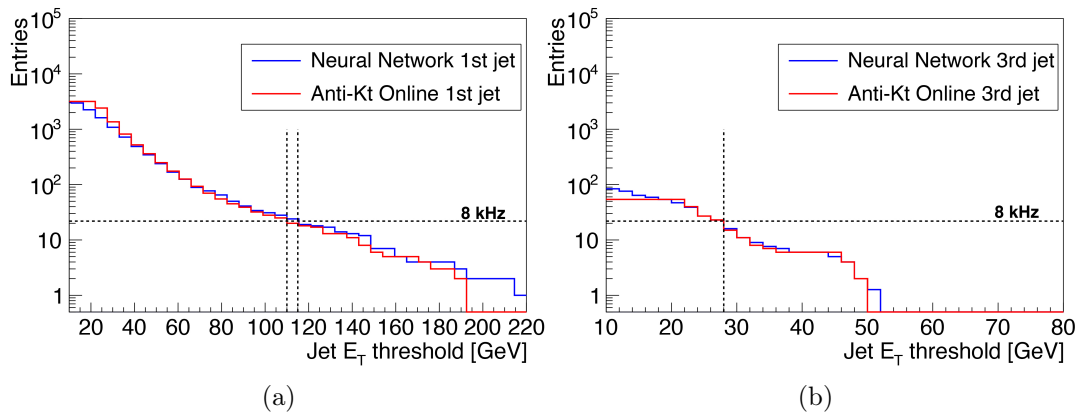


Figure A.10: (a) Single-jet trigger rates and (b) 3-jet trigger rates for the neural network adapted to Phase-2 ACAP resources with global calorimeter input and anti- k_t online.

Bibliography

- [1] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. ISSN: 1435-568X. DOI: [10.1007/BF02551274](https://doi.org/10.1007/BF02551274). URL: <https://doi.org/10.1007/BF02551274>.
- [2] Paul Langacker. *The standard model and beyond*. Series in High Energy Physics, Cosmology and Gravitation. Boca Raton, FL: Taylor and Francis, 2010. URL: <https://cds.cern.ch/record/1226768>.
- [3] Wikimedia. *MissMJ, Nasfarley88 and others*. https://upload.wikimedia.org/wikipedia/commons/0/00/Standard_Model_of_Elementary_Particles.svg. [Online; accessed 11-July-2018]. 2018.
- [4] Bo Andersson, Sandipan Mohanty, and Fredrik Soderberg. “Recent developments in the Lund model”. In: *36th Annual Winter School on Nuclear and Particle Physics (PINP 2002) and 8th St. Petersburg School on Theoretical Physics St. Petersburg, Russia, February 25-March 3, 2002*. 2002. arXiv: [hep-ph/0212122](https://arxiv.org/abs/hep-ph/0212122) [[hep-ph](https://arxiv.org/abs/hep-ph/0212122)].
- [5] Torbjorn Sjostrand, Stephen Mrenna, and Peter Z. Skands. “A Brief Introduction to PYTHIA 8.1”. In: *Comput. Phys. Commun.* 178 (2008), pp. 852–867. DOI: [10.1016/j.cpc.2008.01.036](https://doi.org/10.1016/j.cpc.2008.01.036). arXiv: [0710.3820](https://arxiv.org/abs/0710.3820) [[hep-ph](https://arxiv.org/abs/0710.3820)].
- [6] Thomas Schörner-Sadenius, ed. *The Large Hadron Collider*. Berlin: Springer, 2015. ISBN: 9783319150000, 9783319150017. DOI: [10.1007/978-3-319-15001-7](https://doi.org/10.1007/978-3-319-15001-7). URL: <http://www.springer.com/gb/book/9783319150017>.
- [7] Cinzia De Melis. “The CERN accelerator complex. Complexe des accélérateurs du CERN”. In: (2016). General Photo. URL: <https://cds.cern.ch/record/2119882>.
- [8] Joao Pequeno. “Computer generated image of the whole ATLAS detector”. 2008. URL: <https://cds.cern.ch/record/1095924>.
- [9] G. Aad et al. “The ATLAS Inner Detector commissioning and calibration”. In: *Eur. Phys. J. C* 70 (2010), pp. 787–821. DOI: [10.1140/epjc/s10052-010-1366-7](https://doi.org/10.1140/epjc/s10052-010-1366-7). arXiv: [1004.5293](https://arxiv.org/abs/1004.5293) [[physics.ins-det](https://arxiv.org/abs/1004.5293)].
- [10] G. Aad et al. “Readiness of the ATLAS Tile Calorimeter for LHC collisions”. In: *Eur. Phys. J. C* 70 (2010), pp. 1193–1236. DOI: [10.1140/epjc/s10052-010-1508-y](https://doi.org/10.1140/epjc/s10052-010-1508-y). arXiv: [1007.5423](https://arxiv.org/abs/1007.5423) [[physics.ins-det](https://arxiv.org/abs/1007.5423)].

- [11] G. Aad et al. “Readiness of the ATLAS Liquid Argon Calorimeter for LHC Collisions”. In: *Eur. Phys. J. C* 70 (2010), pp. 723–753. DOI: [10.1140/epjc/s10052-010-1354-y](https://doi.org/10.1140/epjc/s10052-010-1354-y). arXiv: [0912.2642](https://arxiv.org/abs/0912.2642) [physics.ins-det].
- [12] Joao Pequeno. “Computer Generated image of the ATLAS calorimeter”. 2008. URL: <https://cds.cern.ch/record/1095927>.
- [13] P Puzo. “ATLAS calorimetry”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 494.1 (2002). Proceedings of the 8th International Conference on Instrumentation for Colliding Beam Physics, pp. 340–345. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/S0168-9002\(02\)01490-0](https://doi.org/10.1016/S0168-9002(02)01490-0). URL: <http://www.sciencedirect.com/science/article/pii/S0168900202014900>.
- [14] Morad Aaboud et al. “Measurement of the photon identification efficiencies with the ATLAS detector using LHC Run-1 data”. In: *Eur. Phys. J. C* 76.12 (2016), p. 666. DOI: [10.1140/epjc/s10052-016-4507-9](https://doi.org/10.1140/epjc/s10052-016-4507-9). arXiv: [1606.01813](https://arxiv.org/abs/1606.01813) [hep-ex].
- [15] Chris Meyer. “The ATLAS Tile Calorimeter Calibration and Performance”. In: *EPJ Web Conf.* 60 (2013), p. 20051. DOI: [10.1051/epjconf/20136020051](https://doi.org/10.1051/epjconf/20136020051). arXiv: [1310.2945](https://arxiv.org/abs/1310.2945) [physics.ins-det].
- [16] A. Artamonov et al. “The ATLAS forward calorimeters”. In: *JINST* 3 (2008), P02010. DOI: [10.1088/1748-0221/3/02/P02010](https://doi.org/10.1088/1748-0221/3/02/P02010).
- [17] K Ishii. “The ATLAS muon spectrometer”. In: *PoS hep2001* (2001), p. 253. URL: <http://cds.cern.ch/record/855733>.
- [18] “Performance of the ATLAS Trigger System in 2015. Performance of the ATLAS Trigger System in 2015”. In: *Eur. Phys. J. C* 77.CERN-EP-2016-241. 5 (2016), 317. 76 p. URL: <https://cds.cern.ch/record/2235584>.
- [19] E. F. Eisenhandler. “ATLAS Level-1 Calorimeter Trigger Algorithms”. In: (2004).
- [20] Reinhard Schwienhorst. *The Phase-1 Upgrade of the ATLAS First Level Calorimeter Trigger*. Tech. rep. ATL-DAQ-PROC-2015-047. 01. Geneva: CERN, 2015. URL: <https://cds.cern.ch/record/2062021>.
- [21] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. “The Anti-k(t) jet clustering algorithm”. In: *JHEP* 04 (2008), p. 063. DOI: [10.1088/1126-6708/2008/04/063](https://doi.org/10.1088/1126-6708/2008/04/063). arXiv: [0802.1189](https://arxiv.org/abs/0802.1189) [hep-ph].
- [22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [23] Saleh Albelwi and Ausif Mahmood. “A Framework for Designing the Architectures of Deep Convolutional Neural Networks”. In: *Entropy* 19.6 (2017). ISSN: 1099-4300. DOI: [10.3390/e19060242](https://doi.org/10.3390/e19060242). URL: <http://www.mdpi.com/1099-4300/19/6/242>.
- [24] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning Deconvolution Network for Semantic Segmentation”. In: *CoRR* abs/1505.04366 (2015). arXiv: [1505.04366](https://arxiv.org/abs/1505.04366). URL: <http://arxiv.org/abs/1505.04366>.

- [25] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *CoRR* abs/1609.04836 (2016). arXiv: [1609.04836](https://arxiv.org/abs/1609.04836). URL: <http://arxiv.org/abs/1609.04836>.
- [26] K. A. Olive et al. “Review of Particle Physics”. In: *Chin. Phys.* C38 (2014), p. 090001. DOI: [10.1088/1674-1137/38/9/090001](https://doi.org/10.1088/1674-1137/38/9/090001).
- [27] Dan Green, ed. *At the leading edge*. Singapore: World Scientific, 2010. ISBN: 9789814304672. URL: <http://www.worldscibooks.com/physics/7349.html>.
- [28] P. Amaral et al. “Hadronic shower development in iron scintillator tile calorimetry”. In: *Nucl. Instrum. Meth.* A443 (2000), pp. 51–70. DOI: [10.1016/S0168-9002\(99\)01020-7](https://doi.org/10.1016/S0168-9002(99)01020-7). arXiv: [hep-ex/9904032](https://arxiv.org/abs/hep-ex/9904032) [hep-ex].
- [29] S. Agostinelli et al. “GEANT4: A Simulation toolkit”. In: *Nucl. Instrum. Meth.* A506 (2003), pp. 250–303. DOI: [10.1016/S0168-9002\(03\)01368-8](https://doi.org/10.1016/S0168-9002(03)01368-8).
- [30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: [1505.04597](https://arxiv.org/abs/1505.04597). URL: <http://arxiv.org/abs/1505.04597>.
- [31] Inc. Xilinx. *Xilinx Unveils Revolutionary Adaptable Computing Product Category*. <https://www.xilinx.com/news/press/2018/xilinx-unveils-revolutionary-adaptable-computing-product-category.html>. 2018.

Acknowledgements

An dieser Stelle möchte ich mich herzlich bei allen bedanken, die mich während meines Studiums und vor allem im letzten Jahr beim Erstellen dieser Arbeit unterstützt haben.

Zunächst geht ein großer Dank an Prof. Volker Büscher, der mir es ermöglicht hat, in seiner Arbeitsgruppe an diesem spannenden und interessanten Thema arbeiten zu dürfen. Zudem danke ich Prof. Michael Wurm, dass er sich dazu bereit erklärt hat, diese Arbeit als Zweitgutachter zu bewerten.

Besonderer Dank gilt Christian Schmitt für die intensive persönliche Betreuung und das Beantworten zahlreicher Fragen im Laufe des letzten Jahres. Seine Expertise und stets sehr guten Ratschläge haben deutlich zum Gelingen dieser Arbeit beigetragen.

Ebenfalls bedanken möchte ich mich bei den vier Jungs aus meinem Büro für die vielen informativen physikalischen und zum Teil weniger informativen, aber dafür sehr unterhaltsamen, nicht-physikalischen Unterhaltungen sowie bei allen aus dem Higgs-Büro, die immer ein offenes Ohr für Fragen hatten.

Mein größter Dank geht an meine Eltern. Sie haben mich immer unterstützt und waren immer für mich da. Ohne sie wäre ich nicht da, wo ich heute bin. Vielen Dank dafür!