MASTER'S THESIS

Investigation of robustness of b-Tagging algorithms for the CMS Experiment

by

Annika Stein

Master's Thesis in Physics

submitted to the
Faculty of Mathematics, Computer Science and Natural Sciences
RWTH Aachen University

written under the supervision of
Prof. Dr. rer. nat. Alexander Schmidt
Physics Institute III A

Second examiner
Prof. Dr. rer. nat. Martin Erdmann
Physics Institute III A

October 2021

Abstract

With great success, deep learning as one form of machine learning is utilized for various applications and shows its benefits also in the field of high-energy physics, or more specifically, for jet flavour tagging. At the same time, studies on AI safety show the susceptibility of neural networks, even if the modifications of the inputs are only marginal. Those intriguing, yet concerning properties are analysed in the context of b-tagging, where the chosen architecture replicates the DeepCSV algorithm, which is utilized at the CMS experiment. Investigating the response of the tagger is motivated by the later usage of the outputs in physics analyses, as the values for simulation and data are deviating.

The vulnerability of the neural network will be probed by employing adversarial attacks, which involves the addition of a random Gaussian noise term or the systematic application of the Fast Gradient Sign Method. Investigations of the input shapes as well as the ROC curves show how the aforementioned adjustments influence the features and the performance. The scan of the area under the ROC curve over epoch reveals the tradeoff between model performance and susceptibility. A defense strategy, namely adversarial training, improves the robustness of the classifier and represents a better tradeoff, compared to the basic training. This method constitutes a promising candidate to reduce discrepancies when evaluating the model on simulated events as well as on data.

Zusammenfassung

Deep Learning als eine Form des Maschinellen Lernens wird mit großem Erfolg in verschiedensten Anwendungsgebieten eingesetzt und erweist sich auch in der Hochenergiephysik, genauer beim Jet Flavour Tagging, als hilfreich. Gleichzeitig zeigen Studien zum Thema AI safety, wie anfällig neuronale Netze reagieren, selbst wenn nur geringfügige Änderungen der Eingaben vorgenommen werden. Jene verblüffenden, gar beunruhigenden Eigenschaften werden am Beispiel des b-Taggings analysiert, wobei die Architektur den DeepCSV Algorithmus nachbildet, der beim CMS Experiment zum Einsatz kommt. Die spätere Verwendung der Ausgaben für physikalische Analysen gibt Anlass zur Untersuchung des Ansprechverhaltens des Taggers, da abweichende Werte für simulierte und vom Detektor gemessene Daten beobachtet werden.

Die Anfälligkeit des neuronalen Netzes wird unter Verwendung von Adversarial Attacks (zu deutsch etwa „feindlichen Angriffen“) untersucht werden, was die Addition eines zufälligen, Gauß-verteilten Rauschens oder die systematische Anwendung der Fast Gradient Sign Method beinhaltet. Untersuchungen der Eingabeverteilungen sowie der ROC-Kurven zeigen, wie sich die zuvor genannten Veränderungen auf die Merkmale und die Modell-Performance auswirken. Das Aufzeichnen der Fläche unter der ROC-Kurve als Funktion der Epoche legt den Kompromiss zwischen Performance und Anfälligkeit offen. Eine Abwehrstrategie, das sogenannte Adversarial Training, erhöht die Robustheit des Klassifizierers und stellt im Vergleich zum gewöhnlichen Training eine bessere Kompromisslösung dar. Diese Methode liefert einen vielversprechenden Ansatz, um Unstimmigkeiten bei der Evaluation des Modells mit simulierten Events und Daten zu reduzieren.

Contents

1 Introduction

1.1 Motivation and general concepts

Modern machine learning techniques, especially deep learning [1], have grown in popularity for various applications. Ranging from everyday life to the fundamental sciences, deep neural networks are used to approach difficult tasks of which classification problems are only one example [2].

The field of high-energy physics (HEP) is no exception, and strongly profits from newly developed algorithms that contribute to more precise results and better sensitivity. At the Compact Muon Solenoid (CMS), an experiment at the Large Hadron Collider (LHC) at CERN, deep learning has become an important ingredient of several reconstruction steps and analysis procedures. These modern machine learning approaches reveal their potential especially for high-dimensional, large data sets, being able to find patterns in complex structures or to identify rare signals in background-dominated regions. It is expected that the relevance of new deep learning technologies will increase, with the era of the High-Luminosity LHC (HL-LHC) approaching [3].

With the recent advances in deep learning, the necessity to investigate the robustness of the models intensifies as well. Especially the high-dimensional data, which is characteristic for HEP, might lead to systematic uncertainties when provided to deep neural networks [4]. Previous studies report severe impacts on the performance of neural networks with small modifications of the inputs only [5]. Careful exploration of the susceptibility is necessary to examine how concerning these "intriguing properties of neural networks" [6] are in practice.

In particular, this thesis applies methods from AI safety [7] to jet heavy-flavour tagging at CMS [8], with a focus on b-tagging algorithms. This extends the prior work carried out in Ref. [9] and is motivated by Refs. [4] and [10]. Algorithms that classify jets based on the flavour of their initiating particle (a quark or gluon) will be examined [8, 11], probing the tradeoff between performance and robustness. For these studies, the underlying motivation is the possibility of detector effects or subtle mismodellings in simulated events that could be invisible to typical validation methods [4].

Several strategies to artificially manipulate the input data and to investigate the impact on the performance of the tagging algorithm will be applied throughout this study. Additionally, a defense strategy that aims at reducing the susceptibility will be tested, with the ultimate goal of learning less simulation-specific features to achieve better generalization to real detector data [4].

The standard model of particle physics and theoretical aspects related to jets are summarized in Section 1.2. A brief technical introduction to the CMS experiment at the LHC follows in Section 1.3. The basic concepts of flavour tagging and deep learning are discussed in Sections 1.4 and 1.5. This introductory chapter concludes with an explanation of AI safety in Section 1.6. In Chapter 2, the implementation of the classifier and its training procedure are documented. The nominal performance is examined in Chapter 3, whereas Chapter 4 investigates the model's robustness to mismodelling. Concluding remarks on the achieved results and an outlook to possible next steps are presented in Chapter 5.

1.2 Theoretical background

1.2.1 The standard model of particle physics

The standard model (SM) of particle physics describes the fundamental particles and the forces between them in a quantum field theory (QFT), which merges quantum mechanics (QM) and special relativity (SR). Among the four fundamental forces are the strong, the weak, and the electromagnetic force, these are included in the SM with their respective theoretical frameworks called quantum chromodynamics (QCD), quantum flavourdynamics (QFD) and quantum electrodynamics (QED). Gravity constitutes an exception, which is described by the general theory of relativity and does not play an important role for the small scales investigated in particle physics [12].

In the SM, elementary particles emerge from excitations of fields and can be categorized based on the interactions and quantum numbers. Particles with half-integer spin are called fermions, whereas those particles with integer spin are called bosons. Figure 1.1 lists the fermions (fundamental matter particles) and bosons (gauge bosons (force carriers) and the Higgs boson (scalar)) along with their mass, charge and spin; antiparticles of the fermions are not listed and would show up with opposite charges / quantum numbers with flipped signs, but same mass. The fermions can be subdivided into leptons and quarks, these can be separated into

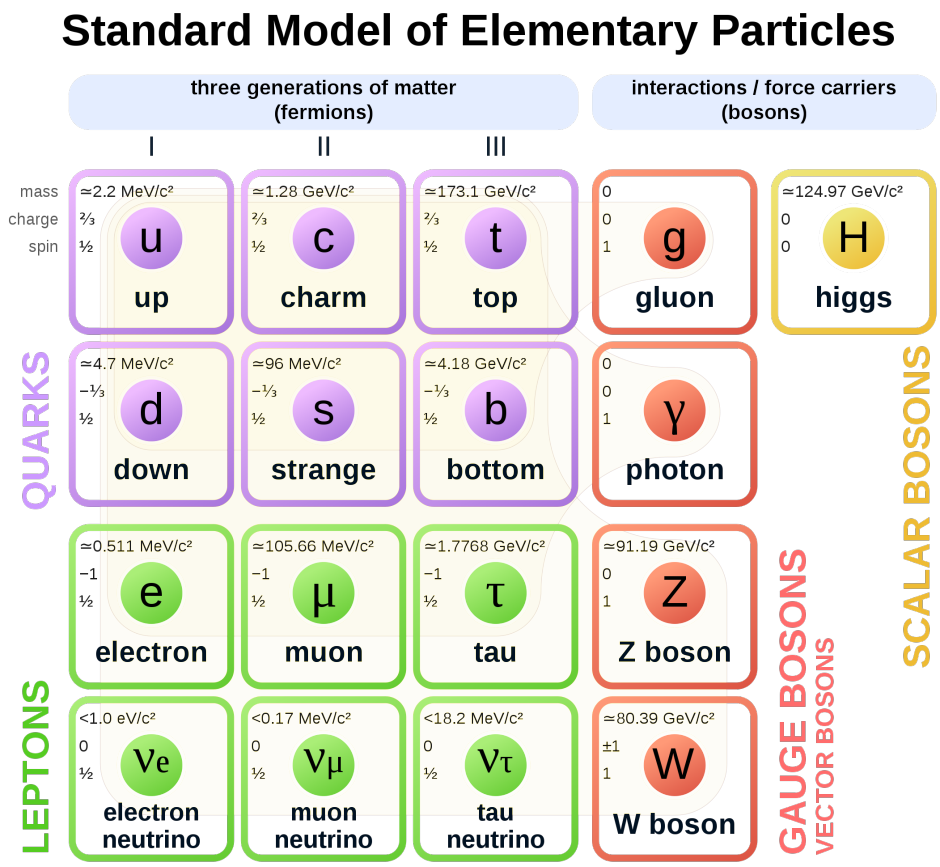

Figure 1.1: The particles in the standard model, including information about the mass, charge and spin of fermions and bosons [13]. Charges are denoted in units of the elementary charge e, the spin is stated in units of $\hbar$. For all subsequent occurrences, the aforementioned simplification will be used and the constants c (speed of light) and $\hbar$ (reduced Planck constant) will be replaced in accordance with the natural system of units where $c = \hbar = 1$.

three generations, where the particles differ by mass. In each lepton generation, there is a doublet consisting of charged leptons ($\ell^{\pm}_{\mathrm{L\,or\,R}}$, interacting via the electromagnetic or weak interaction) and a singlet (neutrinos: ν_{L}, interacting only weakly). For the quarks (q), the three generations are each build up of an up type quark with electric charge $q = +2/3$ and a down type quark ($q = -1/3$), with which they participate in the electromagnetic interaction. Quarks also carry a weak and a colour charge, which means that they interact via the weak and strong interaction. Here, the term colour refers to a quantum number with possible values red, green or blue, which had to be introduced to explain the discovery of the Δ^{++} resonance (otherwise, the existance of this baryon (acting as a fermion) that contains three up quarks would violate the Pauli exclusion principle) [12].

The Lagrange density $\mathcal{L}_{\mathrm{SM}}$ associated with the SM obeys the principle of local gauge symmetry under the gauge transformation induced by the direct product of the symmetry groups $SU(3) \times SU(2) \times U(1)$. In this representation, $SU(3)$ is related to QCD, mediated by eight gluons (g), and acts on colour charge; $SU(2)$ is related to the weak interaction, mediated by the $\mathrm{W}^{\pm}$ and Z bosons, acting on flavour and $U(1)$ is related to QED, where photons (γ) mediate the exchange of electric charge [12]. The Higgs boson is not related to any force in the literal sense; instead it can be interpreted as an excitation of the Higgs field and is connected to spontaneous symmetry breaking in the electroweak sector ($SU(2) \times U(1)$). It generates the fermion masses via Yukawa interactions (couplings proportional to the fermion mass m_f) and the Higgs mechanism solves the problem that in the unified electroweak model, $\mathrm{W}^{\pm}$ and Z bosons should be massless due to local gauge invariance, while observations have shown that their masses are non-zero [12].

1.2.2 Fragmentation, hadronization and jets

In proton-proton collisions (see Fig. 1.2), initially coloured quarks and gluons are produced

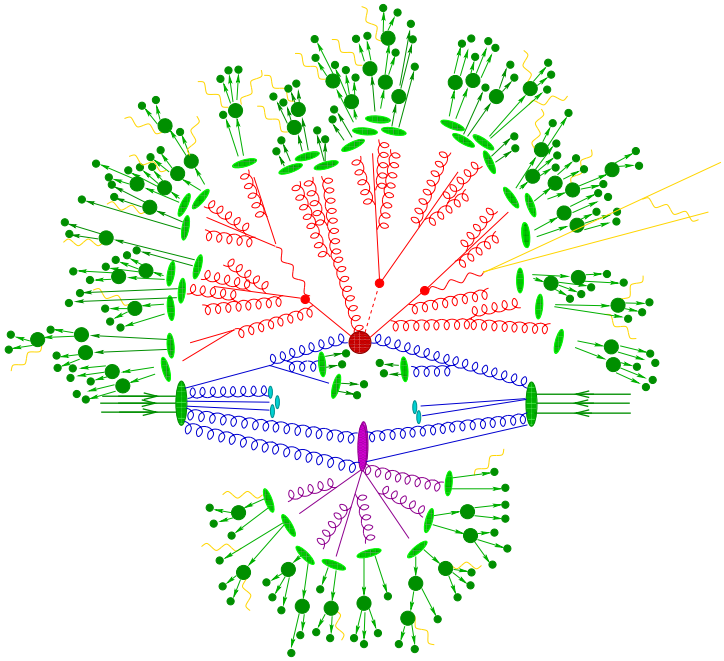

Figure 1.2: Sketch of a hadron collision, resulting from a Monte Carlo event generator [14]. Hard scattering is visualized in the center by a red "blob", a parton shower follows (red). Hadronization is depicted with green blobs, followed by dark green spots that represent hadron decays. Additionally, the purple area shows a secondary hard scattering event (underlying event) [14, 15]. Yellow lines represent soft photon radiation [14].

abundantly as a consequence of the strong interaction of partons inside the proton. When the initially free quark-antiquark pairs separate, the accompanying colour field stores enough energy to form new $q\bar{q}$ pairs (e.g. string model [15, 16]). In a cascade-like manner, governed by quantum chromodynamics, fragmentation (parton showering / splitting) sets in, which creates new particles from the vacuum [12].

Due to colour confinement, quarks and gluons hadronize to colour-neutral composite particles, that is, to hadrons [17, 12]. This imposes some conditions: colour-neutrality can be achieved by forming hadrons out of (valence) quarks by taking e.g.

- three quarks (each with a different colour) — baryons $qq'q''$ (similar with antiquarks),
- a quark and an antiquark that carry a colour and the corresponding anticolour — mesons $q\bar{q}'$,
- combinations of these, like tetraquarks ($q\bar{q}'q''\bar{q}'''$), pentaquarks (e.g. $qq'q''q'''\bar{q}''''$) or more complicated settings,

where the first two cases are most relevant and well-known [18, 12]. As a consequence of this hadronization process (which can only be modelled phenomenologically) and due to subsequent decays of the hadrons, a collimated stream of particles arises [12, 15]. These particles (e.g. charged or neutral hadrons, but also neutrinos and charged leptons from weak decays) leave a cone shaped trace, aligned with the original flight direction of the initiating parton, which gives rise to the term jet [12, 15]. The properties of jets are revisited in Section 1.4.

1.3 The CMS experiment at the LHC

1.3.1 The Large Hadron Collider

With its circumference of $27\,\mathrm{km}$, the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN[1]) near Geneva, at the border of France and Switzerland, is currently the world's largest hadron ring collider [19, 20, 21]. The tunnel, which formerly hosted the Large Electron Positron Collider (LEP), is situated up to $175\,\mathrm{m}$ below ground (depending on the exact geographical environment) and is reachable by several access shafts [22]. In the primary mode of operation, protons are accelerated in bunches up to a center-of-mass energy of $13\,\mathrm{TeV}$ with the help of several pre-accelerators that fill the final stage, that is, the LHC [21, 22]. This value ($\sqrt{s} = 13\,\mathrm{TeV}$) is valid for 2017 (being part of Run 2 from 2015–2018), the year to which the samples used in this thesis correspond to. The quoted value marks the highest center-of-mass energy achieved in a laboratory, which is possible due to (small selection): accelerating cavities, vacuum tubes and a complex magnet configuration that features (among higher order corrections) dipoles for bending and quadrupoles for focusing, which all involve sophisticated engineering [22]. The counter-rotating proton bunches are brought to collision at the interaction points [19], where the four main experiments CMS (Compact Muon Solenoid) [24], ATLAS (A Toroidal LHC Apparatus) [25], ALICE (A Large Ion Collider Experiment) [26] and LHCb (Large Hadron Collider beauty) [27] are located.

After only two years of data taking in Run 1 (2010–2012), the CMS and ATLAS Collaborations announced the discovery of the long-sought Higgs boson in 2012 [28, 29]. This major achievement completes the experimental searches for all fundamental SM particles with the last remaining missing piece. Ongoing studies with experiments at the LHC concentrate on the improvement of SM measurements and the properties of the particles therein, or search for physics beyond the SM.

Reaching the design value of $\sqrt{s} = 14\,\mathrm{TeV}$ is targeted for the next phase of LHC operation

[1] *CERN: Conseil européen pour la recherche nucléaire*

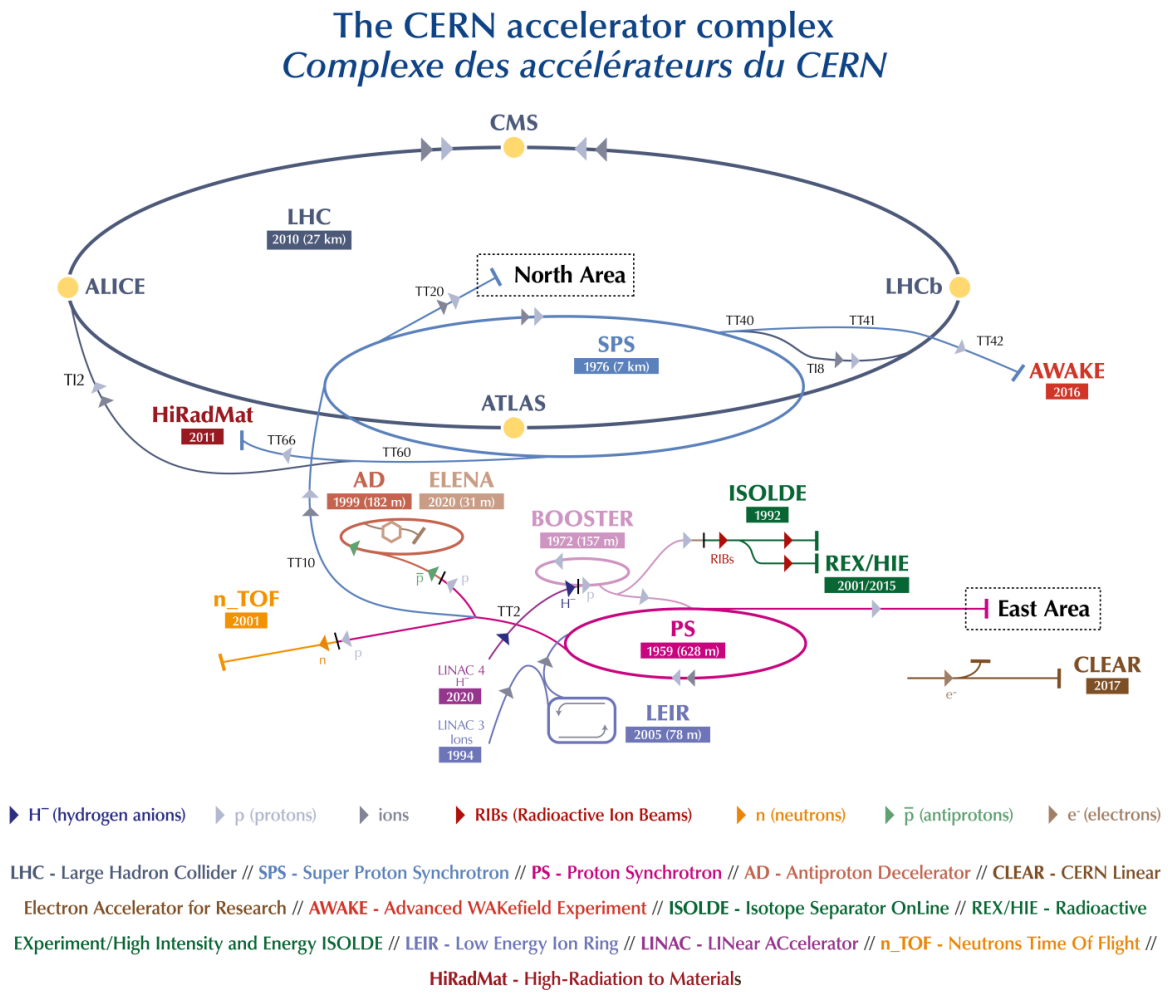

Figure 1.3: Sketch of the LHC and its experiments at the CERN accelerator complex [23].

(Run 3), to be started in 2022, after ongoing upgrades of the accelerator and detector components [19, 30, 31, 32]. Long-term research and development activities also focus on the High-Luminosity LHC [33, 30, 34], which will, as the name states, increase the achievable luminosity (e.g. from the current / nominal instanteneous luminosity of $1 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ to $\approx 7.5 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ [30]). Multiplied with individual cross sections, this results in a higher number of events, however, at the price of larger data sets, a more difficult reconstruction as well as the demand for improved radiation hardness of the detector components, and in analyses that need to deal with unwantedly delivered pileup [33, 30, 34, 3]. In the planned setup, the average pileup μ (or the number of events per bunch crossing) would increase from 27 to around 200 [30].

1.3.2 The Compact Muon Solenoid

CMS, a multi-purpose detector, is located $100\,\mathrm{m}$ underground at one of the interaction points where counter-rotating proton bunches are brought to collision [24]. Its comparatively compact dimensions of $21.6\,\mathrm{m}$ in length and $14.6\,\mathrm{m}$ in diameter, with a weight of $12\,500\,\mathrm{t}$ [24] justify the first letter in the abbreviation, when put in contrast with ATLAS [25, 22].

Detector components Starting with the innermost component, the detector utilizes a silicon pixel and strip tracker close to the beam pipe, an electromagnetic calorimeter (ECAL) made of lead tungstate crystals, a hadron calorimeter (HCAL) that consists of brass and scintillator material, a superconducting solenoid and a muon system of gas-ionization chambers that are enclosed in a steel flux-return yoke [24]. The detection modules are part of a barrel and two endcap sections, as visualized in Fig. 1.4. Some of the working principles and

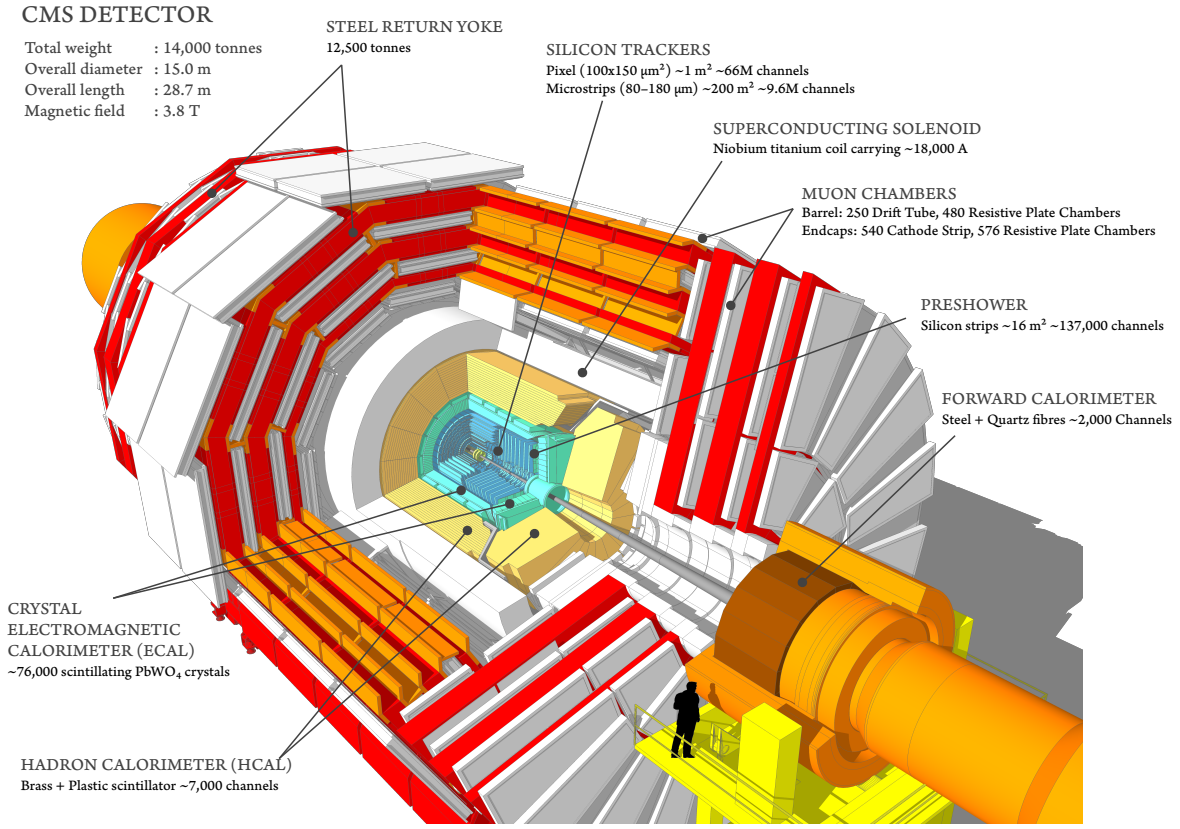

Figure 1.4: The CMS detector with its components, visualized with a 3D model [35]. The denoted dimensions differ slightly from the original configuration [24].

functions of those layers are discussed below.

- **Tracker:** The tracking system consists of silicon pixels, of which some are part of currently four barrel layers since the last Phase-1 upgrade, and of silicon strip modules [24, 31]. Both types of elements provide high-efficiency for the detection of charged particles up to pseudorapidities of $|\eta| < 2.5$ (this is the tracker acceptance) [24, 36, 37].
- **ECAL:** An electromagnetic shower is created (mainly) when electrons and photons interact with the lead tungstate crystals. The scintillator material subsequently emits scintillation light that can be detected with photodetectors. As a result, the energy of the originally absorbed particle can be measured [24, 38].
- **HCAL:** In this sampling calorimeter, brass alloy absorbers and plastic scintillators are placed alternatingly. It plays an important role in determining the energy of the hadronic component (e.g. jet energy scale) and also helps in the indirect determination of the missing energy (e.g. neutrinos that do not interact and are not charged) [24, 39].
- **Solenoid:** The solenoid that consists of superconducting coils is one of the core components of CMS and provides a strong and homogeneous magnetic field of $3.8\,\mathrm{T}$ inside its internal volume with $6\,\mathrm{m}$ diameter. With the magnetic field, it is possible to obtain the charge and momentum of charged particles, as their tracks are bent according to the

Lorentz force. To not affect the measurement of the deposited energy with additional absorber material, the solenoid is positioned outside both calorimeters [24].

- **Muon chambers and return yoke:** Gas-ionization chambers are placed in the outer layer of the detector, embedded into the steel flux-return yoke, where the orientation of the magnetic field is reversed with respect to the inner part [24]. The drift tube (DT) chambers in the barrel part, cathode strip chambers (CSC) for the endcaps, and resistive plate chambers (RSC, for barrel and endcaps) contribute to the detection of muons, a task where CMS excels at, hence the second word behind its acronym [24, 40].

Trigger system A two-tiered trigger system is used to reduce the event rate from the originally expected collision frequency of $40\,\mathrm{MHz}$ (which corresponds to a $25\,\mathrm{ns}$ bunch spacing, the time between each bunch crossing), because the full information produced would otherwise exceed the storage capacities [24, 41]. The so called level-1 (L1) trigger operates directly on specialized hardware processors and imposes (weak) conditions to select events at a rate of $100\,\mathrm{kHz}$, using information from the calorimeters and muon detectors. With the subsequent high-level trigger (HLT), the event rate is further reduced to approximately $1\,\mathrm{kHz}$ by running a simplified, fast version of the full reconstruction software. There, rather complex requirements and tracker information can be taken into account to save potentially interesting events for an analysis on tape [24, 41].

Coordinate system To describe the event and its reconstructed particles inside the detector, CMS utilizes a special right-handed coordinate system with its origin at the primary collision point [24]. The x axis points horizontally inward to the center of the LHC, the y axis points vertically upwards and the z axis points along the beam direction. With these definitions, the x-y plane is transverse to the beam axis. An azimuthal angle ϕ with $\phi \in [-\pi, \pi]$ is defined in that transverse plane, where $\phi = 0$ coincides with the positive x axis. Another quantity used to describe positions in the transverse plane, namely the coordinate r, is measured from the primary interaction vertex of the colliding protons, pointing radially outwards [24].

The polar angle $\theta \in [0, \pi]$ is measured in relation to the positive z axis. The pseudorapidity η is defined as a function of the angle θ between the beam axis and the trajectory of a particle in the following way:

$$\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right). \tag{1.1}$$

In the relativistic limit, the pseudorapidity η approximates the rapidity y, defined as

$$y = \frac{1}{2}\ln\left(\frac{E + p_\mathrm{z}}{E - p_\mathrm{z}}\right), \tag{1.2}$$

where E refers to the energy, and p_z to the z-component of the momentum of the particle. Differences between rapidities are invariant under Lorentz boosts along the beam direction, and in the special case mentioned above, where $\eta \approx y$ for massless particles, the same holds also for pseudorapidities [12, 15].

Besides negligible inclination angles between colliding proton bunches (i.e. assuming head-on collisions), the initial transverse momenta $p_\mathrm{T} = \sqrt{p_x^2 + p_y^2} = p\sin(\theta)$ of the colliding particles are zero. The longitudinal component, however, can not be exactly known from the beginning, not even in theory, as the protons consist of partons that carry fractions of the total momentum, given by parton density functions [12]. However, rapidity differences are not affected by the fact that the longitudinal boost of the parton-parton system is unknown (*a priori*), as stated above [12].

Object reconstruction Iterative tracking of charged particles is performed using a Kalman filter to find patterns between clusters and hits obtained from the pixel and strip modules [8, 37]. The position of the primary proton-proton interaction vertex (PV) needs to be found and tracks need to be fitted with dedicated vertexing algorithms. Later, when all physics-objects are available, the reconstructed vertex candidate with the largest value of summed physics-object p_{T}^2 will be taken as the PV [8].

Further, the particle-flow (PF) [42] algorithm is comprised of several steps that first of all link information from the different sub-detectors, identify and reconstruct so called PF candidates and then combine the obtained information with the goal of getting precise energy and momentum measurements. Those particle candidates, namely photons, electrons, muons, charged and neutral hadrons, are utilized as inputs for subsequent reconstruction steps that perform tasks like jet clustering [8, 42].

1.4 Jet reconstruction and jet flavour tagging at CMS

1.4.1 Jet reconstruction

From a detector point of view, it is sufficient to think of jets as an observable, collimated stream of colour-neutral, stable particles, i.e. hadrons that deposit their energy, produce hits and that might be accompanied by (soft) leptons [8, 12]. A generalized form of the most common jet clustering algorithms utilizes a sequential procedure that merges the jet's constituents into larger entities, until all particles have been used [43, 44]. Geometrical and kinematic properties are combined into distance measures that can be computed from the transverse momentum $k_{\mathrm{T},i}$ of the i-th particle, its rapidity y_i and azimuthal angle ϕ_i. A pairwise comparison of all constituents i, j is performed iteratively:

$$d_{ij} = \min\left(k_{\mathrm{T},i}^{2p}, k_{\mathrm{T},j}^{2p}\right)\frac{\Delta R_{ij}}{R} \qquad \text{and} \qquad d_i = k_{\mathrm{T},i}^{2p}\,, \tag{1.3}$$

$$\text{where } \Delta R_{ij} = \sqrt{(y_i - y_j)^2 + (\phi_i - \phi_j)^2} \text{ and } p = \begin{cases} 1, & \text{for } k_{\mathrm{T}}; \\ 0, & \text{for Cambridge / Aachen;} \\ -1, & \text{for anti-}k_{\mathrm{T}}. \end{cases} \tag{1.4}$$

If the minimal distance corresponds to a single d_i, this object is taken to be a jet and will not be used for the next iteration; if instead a pairwise distance d_{ij} is the smallest one, those two constituents are combined by summing their four-momentum. Then, the algorithm proceeds with the updated collection of physics-objects or pseudojets. Here, the anti-k_{T} jet clustering algorithm [43] with a jet cone size of $R = 0.4$ is applied.

Secondary vertices that result from displaced tracks and which play an important role for heavy-flavour tagging (see next section) are reconstructed with the inclusive vertex finder (IVF) [37, 8]. The removal of pileup contributions is handled with the pileup per particle identification (PUPPI) algorithm [45], with which contaminations of particles that arise from other primary vertices can be subtracted [8]. Various steps to apply a jet energy calibration need to be followed prior to an analysis, because the energy obtained from the sum of the reconstructed jet constituents does not match the true jet energy [46, 8]. More specifically, one would need to take contributions due to pileup, the response of the calorimeter, missing energy of neutrinos related to the jet's flavour (b/c) and disagreements between data and simulation into account, but for the training of the classifiers in this study, only the raw quantities are used [8, 46].

1.4.2 Jet flavour tagging

Distinguishing heavy- from light-flavour Jets

As already described in the previous section, the showering / fragmentation of partons with their subsequent hadronization, followed by characteristic decay chains, results in a distinct cone of collimated particles [12], known as a jet (see Fig. 1.5). Jet heavy-flavour tagging, or often used synonymously, jet flavour identification, describes the process of distinguishing jets originating from b (bottom) or c (charm) quarks from those that are initiated by light quarks (up / down / strange) or gluons (g) [47, 11]. The term b(c)-tagging will be utilized when the identification focuses on b(c) jets [8, 11]. Top quarks (with their mass of $m_{\mathrm{t}} \approx 173\,\mathrm{GeV}$ being close to the electroweak scale at $v \approx 246\,\mathrm{GeV}$, related to the vacuum expectation value v of the Higgs field) do not hadronize and therefore no top quark-initiated jets are expected [15]. Instead, the top quark directly decays through the weak interaction, mainly via $\mathrm{t} \to \mathrm{Wb}$ [8, 15].

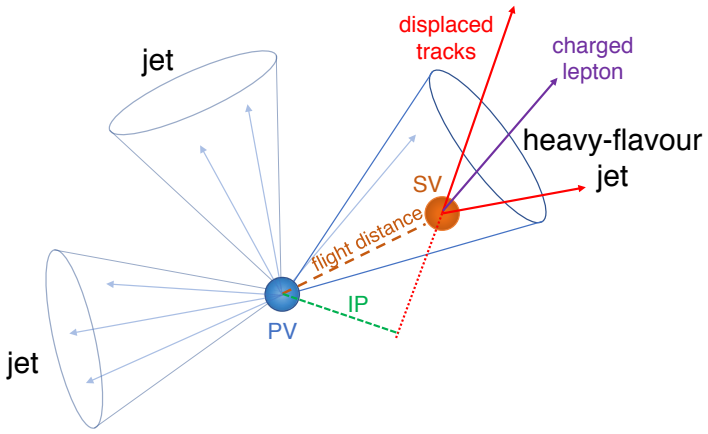

Figure 1.5: Visualization of three jets, emerging from a primary vertex (PV), where distinct properties of the heavy-flavour jet like a secondary vertex (SV) with displaced tracks and a large impact parameter (IP) are highlighted [8].

The key properties of heavy-flavour jets that allow for their discrimination from light-flavour jets are described below.

- The **long lifetimes** of **heavy** b and c hadrons in their rest frames are of the order of $1.5\,\mathrm{ps}$ and $1\,\mathrm{ps}$, respectively. Due to this feature, for example, the displacements of b hadrons (measured in the detector frame for $\sqrt{s} = 13\,\mathrm{TeV}$ proton-proton collisions) typically reach a few mm to one cm, after which a secondary vertex (SV) is expected [8, 15]. There, the decay of a heavy hadron generates charged-particle tracks that are themselves displaced with respect to the primary vertex. For the b hadrons, the dominant decay mode in the "spectator model" involves the transition of a b quark to a c quark and a virtual W boson (that itself decays to a pair of leptons or quarks). Though this model is only approximate, it describes the decays of b hadrons (mesons and baryons containing b quarks) well, as the lifetime differences of different b hadrons scale with the inverse square of the b quark mass [15]. These differences are small, compared to the ones for light-flavour hadrons [15]. By computing the invariant mass of the reconstructed particles that emerge from the SV, an observable that is directly correlated to the mass of the decaying particle can be deduced [8, 48]. Also the "flight distance", which measures the distance between the PV and the SV, offers discriminating power, because it incorporates the lifetime indirectly (the lifetime can be related to a time constant in an exponential decay and the conversion from time to distance can be done with $\gamma c\tau$ [15]).

Measuring the displacement of tracks is done with the help of the impact parameter vector that points from the PV to the point of closest approach. Its length (a distance measure) is called the impact parameter. An important input feature, namely the signed impact parameter, combines this quantity with a sign that is obtained from computing the scalar product between the impact parameter vector and the jet axis [8].

- The relatively **high masses** of b and c quarks ($\approx 4.2\,\mathrm{GeV}$ and $\approx 1.3\,\mathrm{GeV}$, respectively [15]) and the **hard fragmentation** cause the decay products of the initial b or c hadrons to have larger p_T values than other jet constituents, when measured against the jet axis [8].
- With branching fractions of approximately $20\,\%$ ($10\,\%$), **charged leptons**, namely electrons or muons, are generated in the (semi-)leptonic decay chain of b or c hadrons [8, 48, 15]. This allows to exploit additional features in the training of combined taggers and can be used to select events in data to measure the performance of the jet flavour identification algorithms and to calibrate them in different channels [8, 48].

While the properties of b jets differ drastically from those of light-flavour jets, the identification of c jets is particularly difficult, as their properties like mass or lifetime are situated between those of b and light jets, respectively [8, 48]. A more detailed description of the input variables used for this study that provide discrimination power follows in Section 2.2.2.

Usage and development of jet flavour taggers

Identifying the jet flavour plays an important role in various analysis branches exploited by experiments like CMS [8, 47]. For example, the development of the corresponding jet-tagging algorithms goes hand in hand with analyses that focus on the decay of the Higgs boson to bottom quark-antiquark pairs (e.g. observation of $\mathrm{H} \to \mathrm{b\bar{b}}$) [49, 47]. Ongoing searches also target the $\mathrm{H} \to \mathrm{c\bar{c}}$ decay mode, where not only the smaller branching fraction, but also the identification of c jets increases the difficulty [50].

The heavy-flavour jet identification algorithms used at CMS have evolved over time, in complexity and in performance. Solely track based taggers (e.g. jet probability taggers) compute a likelihood that indicates compatibility of tracks to originate from the primary vertex [8]. There, the important variable that helps discriminating between displaced and non-displaced tracks is the signed impact parameter significance. Combined secondary vertex taggers include the information from tracks as well as secondary vertices by utilizing likelihood ratios, multivariate analysis techniques like boosted decision trees, and lately also artificial neural networks [8, 11]. The DeepCSV (Deep Combined Secondary Vertex) tagger [8] is one such example where a fully-connected, deep neural network is trained on several input variables related to the jet, its tracks and secondary vertex properties. Several other algorithms exist, for example, the properties of soft leptons in the jet provide additional discriminating power for c-taggers. Compared to DeepCSV, significantly more input variables, of which many are low-level quantities, are used for the DeepJet algorithm [51], where a more complex structure involves convolutional, recurrent as well as dense neural network components. Another branch of tagger development aims at identifying jets in boosted topologies [8, 47].

For this work, the architecture of DeepCSV will be replicated, as it is still commonly used and sufficiently deep to reveal possible susceptibilities to minimal distortions of the input features. Several technical terms related to deep learning have already been used implicitly, consequently it is now time to delve into the foundations of neural networks and their properties relevant for the following AI safety studies.

1.5 Deep Neural Networks

1.5.1 Basics and terminology

Being an example of machine learning (ML) algorithms, a neural network (NN) is able to solve a task by learning from the data it is presented. Doing so involves approximating a complex function f that maps the inputs to the respective outputs of the network [2].

One common task is to figure out to which out of several categories a given input belongs to. This is called a *classification* problem [2]. For jet flavour tagging this will be the case as well, where formally, neural networks are asked to find the mapping $f : \mathbb{R}^n \to \{1, \dots, k\}$ between some inputs $\vec{x} \in \mathbb{R}^n$ of the jet and the flavour $y \in \{1, \dots, k\}$ from which the jet originates [8]. More generally, the inputs are also called *features* and the flavour can be identified with the *ground-truth class* (also called *label* or *target*) [2, 52].

The building blocks of neural networks are the *neurons* (or *nodes*), which are part of (potentially many) *layers*. Using the term neuron can be interpreted as a reminder that the early developments of neural networks are influenced by neuroscience [2]. What distinguishes a *deep neural network* (DNN) from "shallow" architectures (or for example a boosted decision tree (BDT)) is the connection of many layers with high-dimensionality, each associated with a function of the previous layer's outputs [2, 4]. Chaining different functions increases the depth of the model, which gives rise to the concept of *deep learning*, carried out with the help of *deep feedforward neural networks* (or *multilayer perceptrons*) [2].

In this context, the *input layer* (first layer) contains all nodes that are directly supplied with one of n input variables, the *output layer* (last / final layer) carries predicted values for the k categories (depending on the final activation function, this could directly be a probability for the predicted label $\hat{y}$). Between these two layers, there are typically several *hidden layers* whose parameters need to be learned by the training algorithm in order to arrive at predictions close to the desired labels. A special type of neural network that will be used for this study connects every node of a given layer with all nodes of the previous layer and is called fully-connected neural network (FCNN) [52].

The calculation of the network's output distribution can be formalized by composing linear functions with non-linear functions repeatedly. Obtaining the linear pre-activation for the nodes of a layer l involves multiplying a weight matrix $W_{l,l-1}$ with the outputs of the previous layer $\vec{x}_{l-1}$, as well as adding a bias term $\vec{b}_l$. Every weight matrix contains all the connections between subsequent layers and therefore has the dimension of the product of the number of nodes in layer l and layer $l-1$ [2]. The formula for the pre-activation reads:

$$\vec{x}_l\,' = W_{l,l-1} \cdot \vec{x}_{l-1} + \vec{b}_l. \tag{1.5}$$

This (vectorized) result is plugged into an activation function f_a that introduces non-linearity (the motivation for this is explained in a later paragraph):

$$\vec{x}_l = f_a\left(\vec{x}_l\,'\right) = f_a\left(W_{l,l-1} \cdot \vec{x}_{l-1} + \vec{b}_l\right). \tag{1.6}$$

As mentioned earlier, the network structure allows to connect several such activations to one composite function. Written down with one master equation that describes the whole model, several activation functions $f_a^{(1)}, \dots, f_a^{(d)}$, d being the number of layers, can be used as follows [2, 52]:

$$\begin{aligned}\vec{x}_d &= \text{network output} \\ &= f_a^{(d)}\left(W_{d,d-1} \cdot \left(\dots f_a^{(3)}\left(W_{3,2} \cdot \left(f_a^{(2)}\left(W_{2,1} \cdot \left(f_a^{(1)}\left(W_{1,0} \cdot \vec{x} + \vec{b}_1\right)\right) + \vec{b}_2\right)\right) + \vec{b}_3\right)\dots\right) + \vec{b}_d\right).\end{aligned} \tag{1.7}$$

All weight matrices $W_{l,l-1}$ as well as the bias terms $\vec{b}_l$ are learned parameters, whereas the activation functions $f_a^{(l)}$ are specified ahead of time; $\vec{x}$ are the original inputs and $\vec{x}_d$ are the final outputs of the network. The initialization of the weights at the beginning of the training is therefore the only part where randomness is introduced, otherwise the calculation is purely deterministic. To be able to compare different setups, identical random seeds are set [52].

By using non-linear activation functions for each neuron, the universal approximation theorem [53, 54, 2] ensures that, with a sufficient number of hidden layers, a wide class of functions can be approximated with the neural network, under the assumption that the functions are Borel measurable (well-known examples are continuous functions on a closed and bounded subset of $\mathbb{R}^n$) [2]. This is very helpful when applying the model in a real-world application (for example, to solve problems in particle physics), where linear functions can not be assumed to be sufficient. If the network was composed of linear operations like scaling an input and adding an offset (bias) only, also the complete model would be limited to solely describe linear functional dependencies, because compositions of linear functions remain linear [2, 52].

A measure that compares the model output or *prediction* $\hat{y}$ to the desired truth class y, namely the *criterion* or *loss function*, needs to be evaluated during the training phase of the model [52]. This can be interpreted as an objective function that shows the cost or error rate, which shall be minimized while iterating over the training samples. How this goal can be achieved during the training will be part of the next section. The categorical cross entropy loss function used in for this thesis will be introduced in Section 2.1.2, but the setup and class distribution require slight modifications that are discussed in Sections 2.3.2 and 2.3.3. In general, these basics are revisited in Chapter 2, which provides the detailed implementation.

1.5.2 Training of DNNs

Iterating over the full data set is done several times, all available inputs are presented in every epoch of the training. Each epoch constitutes one execution of the training loop, which will be described in the following. In general, the selected training algorithm splits the data set into minibatches, evaluates the model on those samples inside the minibatch (forward pass), computes the loss and accumulates its gradient with the help of backpropagation, to finally update the model parameters by gradient descent (or similar related gradient-based optimization algorithms). Both the features as well as the labels are given, therefore this procedure is an example of a supervised training algorithm [52, 2].

In more detail, backpropagation starts by computing the gradient of the loss function on the output layer and recursively moves backwards through all layers of the network. For every layer, the gradient of the loss function with respect to the layer's output or activation is taken into account and the gradients on weights and bias terms are calculated subsequently. The backpropagation algorithm itself takes these gradients and propagates each of them to the next, lower-level hidden layer [2]. Thus, backpropagation implements the chain rule of calculus for the neural network. There are dedicated optimizers that make use of the gradients to update the model parameters inside the training loop with the goal to minimize the loss function [52], and the chosen method for this study is further described in Section 2.1.2.

To summarize the implementation, there is an outer loop (iterating over epochs) and an inner loop, iterating over batches, which incorporates the parameter updates. After the execution of the inner loop, the model is also evaluated on unseen data, for validation purposes. How and why this splitting into different data sets is done will be explained in Section 2.2.5.

Assuming that the training of the neural network is already done, the model can be evaluated by forward passes through all connections to obtain a final output. Dedicated evaluation metrics exist with which the performance can be measured. These will play a crucial role when

comparing different training setups or distortion methods. Chapters 3 and 4 will introduce such performance measures and apply them to conduct AI safety studies.

1.6 AI safety

1.6.1 General remarks on AI safety

In a quantitative field like HEP, where extensive simulations are an integral part of physics analyses, it is necessary to understand potential concerns that arise when high-dimensional data is used in conjunction with deep neural networks. Therefore, studying the behaviour of the model in question with respect to the inputs becomes an important strategy to identify issues that could lead to errors when applying the network in an experimental analysis. Especially the transition from "shallow" networks to the usage of deep learning methods has resulted in impressive results that allow precise measurements of rare processes [4].

At the same time, with the higher complexity of the used models, the interpretability suffers from an intransparent, black-box-like structure [55, 6] and questions on the safety of systems that use artificial intelligence (AI) arise [6, 7]. The research topic "AI safety" bundles several problems, the one that is closest to the goals for this thesis is called "robustness to distributional shift" [7, 56]. It is certainly conceivable that subtle mismodellings in the simulation could end up as a possible sign of new physics, or on the contrary, potential new discoveries might be falsely excluded [4]. When evaluating the model on detector data and comparing this with simulation, uncertainties might occur that are related to the specifics of the simulated events (as these have been generated using Monte Carlo (MC) methods that could show software- or algorithm-specific artefacts) [4]. The mismodellings can arise at various steps during the simulation chain, starting with the hard process (matrix element calculation), followed by the subsequent steps that model the parton shower, fragmentation and hadronization, where the perturbation order is limited, and ending with the detector simulation [48]. All these imperfections in the modelling, particularly for variables with high discriminating power like track or SV displacement that are also sensitive to detector alignment, demand the calibration of the discriminator shapes [48] and call for investigations of the tagger response to slightly distorted input data [4]. The approach followed in this thesis does not eliminate possible mismodellings, nor does it provide a definitive *a posteriori* correction, but it helps estimating to what extent tagging efficiency and misidentification rates could be influenced by mismodellings, which in turn affects meausurements that rely on the network's predictions [47, 4].

More specifically, the necessity to conduct AI safety studies is related to the fact that deep neural networks can be particularly sensitive to small perturbances of inputs [6]. The combination of the linear nature of deep neural networks [5], together with the large number of input variables [4] is indicated as an explanation for the susceptibility. The explicit construction of slightly distorted new data points near the original inputs with the goal to confuse the model (*adversarial examples*) can lead to misclassification that results in a deteriorated performance [5, 2]. Applied to computer vision / image recognition, it has been demonstrated that modifications that involve only one pixel are enough to "fool" a neural network [57]. In a more natural language, this means that the classifier that would normally be able to assign the correct class to the input now outputs a wrong category.

In the case of heavy-flavour tagging, *adversarial attacks* are constructed to provide a transfer function that represents potential measurement effects or simulation-specific mismodellings [4] that would normally stay undetected with usual validation methods [4]. As a prototype, adversarial attacks on jets have been implemented in Ref. [10], on which Ref. [9] and this thesis are based.

The investigation could focus on small disturbances only, where the input distributions are only marginally affected, but the performance suffers noticeably. This is the typical case for AI safety studies and applicable to HEP, where huge distortions of the simulated data are excluded via commissioning (see, e.g., Refs. [58, 59]), but correlations could play an important role [4]. This thesis will also probe the response of the network for cases with large distortions of the inputs that would be caught during validation. With the help of performance evaluation measures, this allows to identify robust or susceptible models, an important criterion when choosing flavour tagging algorithms in an analysis.

Different methods to distort the inputs exist, two of them will be explained in Section 1.6.2 and are further applied in this thesis. A technique with the goal to improve the robustness of the classifier, namely *adversarial training*, is introduced in Section 1.6.3.

The questions to be answered by applying various AI safety ideas can be summarized as follows:

- How susceptible are neural networks when adversarial examples are injected?
- Would the differences in the input variables be visible with common validation techniques?
- Can the robustness be improved by applying defense strategies like adversarial training?
- *Outlook:* What could be the implications regarding the application to real detector data?

1.6.2 Strategies to distort inputs

The following explanations present the general concepts behind the chosen techniques to distort the input variables. For a detailed description of the application and some constraints that arise when using those methods for the model in question, the reader is referred to Chapter 4.

Adding Gaussian noise

A non-systematic strategy to create a new, slightly distorted set of inputs randomly shifts the variables by adding a noise term ξ to the original inputs, drawn from a Gaussian distribution [6, 60]. The parameters μ (mean value) and σ (standard deviation) can be specified and allow for a range of small or more severe adjustments of the inputs. Formally, this technique can be described in the following way [2, 5]:

$$x_{\text{noise}} = x_{\text{raw}} + \xi \qquad \text{where} \qquad \xi \sim \mathcal{N}\left(\mu, \sigma^2\right) \qquad \text{with} \qquad P(\xi) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\xi-\mu}{\sigma}\right)^2}. \quad (1.8)$$

As will be described in Section 2.2.5, the inputs to be distorted are scaled to have a standard deviation of one and are centered at zero, such that adding the noise term without additional modifications is possible.

The effect of this distortion is shown in Fig. 1.6. Only one arbitrary input x_i has been chosen for the visualization, and the displayed loss function is just an illustration. A special feature of this method is that the distortion does not know about the model or the behaviour of the loss function with modified inputs. Therefore, there is no preferred direction into which the inputs are shifted. The outcome when looking at the loss function is not necessarily bad, it depends on the steepness of the loss surface surrounding the modified inputs (Ref. [62] relates the susceptibility to the curvature of the decision boundary of the classifier). This technique can be interpreted as a smearing of the inputs and is not really an example for an adversarial attack. On the other hand, this case is of interest because simulation-specific artefacts or detector effects do not necessarily act as a "demon" that wants to attack the model, but might appear as

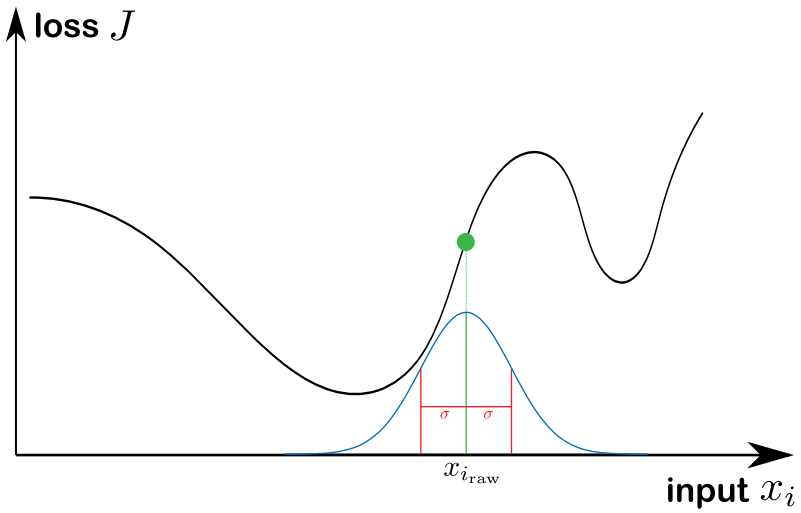

Figure 1.6: Visualization of the random shift of inputs by adding a Gaussian noise term. With the slight distortion based on the blue probability distribution, the formerly green raw datapoint is shifted and the corresponding loss is modified. The change of the loss function with respect to the distorted inputs can go in either direction. For all considered inputs where this method is applicable, the small additional noise term is drawn from a normal distribution with the same fixed standard deviation σ and in the current setup without any offset ($\mu = 0$). Gaussian distribution adapted from Ref. [61].

random fluctuations [4]. Technically, choosing different parameters for every input dimension is possible, but in the experiments conducted for this study, σ is identical for all variables and $\mu = 0$. It is expected that the performance of the model will decrease slightly, as the smearing reduces the discriminating power between the different categories once the distributions are spread out.

Fast Gradient Sign Method

As has been introduced in Ref. [5], a systematic way to generate adversarial inputs, the so called Fast Gradient Sign Method (FGSM), allows to shift the inputs such that the loss function increases. This will in principle result in a decrease in performance, when the model is evaluated on those new inputs. Using the terminology of Ref. [63], this is a white box attack with full knowledge of the network (architecture and parameters). Studying the susceptibility of neural networks to manipulations of that kind allows to get an idea of how the model would do on real-world data, while being trained with dedicated, simulated samples only [4]. Conceptually, the direction of the steepest increase of the loss function around the raw inputs is computed. Once the direction is known, this vector is multiplied with a (small) limiting parameter ϵ to control the severity of the impact. Then, the raw inputs are shifted by this quantity [5, 2]. It can therefore be seen as a technique to maximally disturb the inputs or maximally confuse the network without necessarily becoming obvious in the input variables. Mathematically, the operator that allows to retrieve the "steepest increase" is the gradient of the loss function with respect to the inputs, of which only the sign is kept to get the correct direction, hence the name of the described attack.

Expressed in a single equation, the FGSM attack generates adversarial inputs x_{FGSM} from raw inputs x_{raw} by computing [5]

$$x_{\text{FGSM}} = x_{\text{raw}} + \epsilon \cdot \text{sgn}\left(\nabla_{x_{\text{raw}}} J(x_{\text{raw}}, y)\right). \tag{1.9}$$

In Eq. (1.9), the loss function is denoted as $J(x_{\text{raw}}, y)$ and reflects the dependency on inputs

(x_{raw}) as well as on targets (y). The loss function does depend on the model parameters as well, but for the FGSM attack, this is irrelevant. Moreover, the FGSM attack can be interpreted as a method that locally inverts the approach of gradient descent by performing a gradient ascent with the loss function, but in the input space [5, 63, 64].

The corresponding visualization is shown in Fig. 1.7, however, for didactic reasons with one input variable x_i only. In reality, this method has to be applied multidimensionally, taking all input dimensions into account.

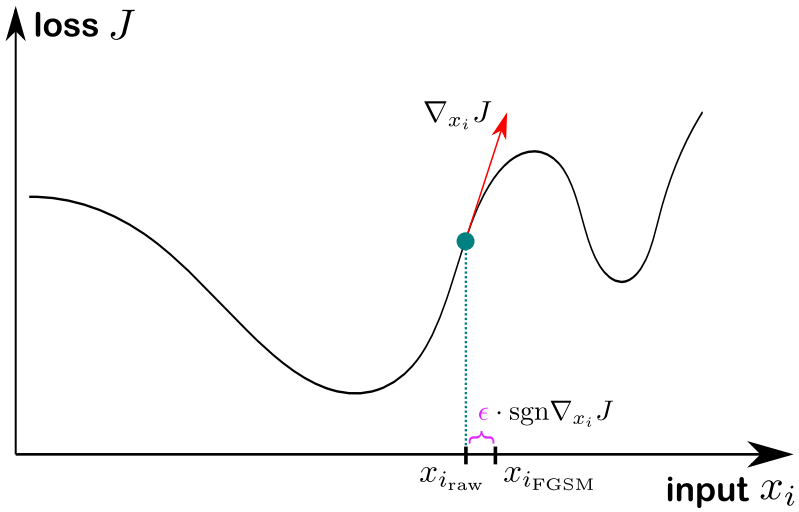

Figure 1.7: Visualization of the generation of adversarial inputs by applying the FGSM attack. The distorted inputs are created in such a way that the loss is maximalized, as explained in the text. Only one input is shown, in the real application, the gradients are computed multidimensionally for all inputs, but each shift uses the same limiting parameter ϵ.

The motivation behind taking the sign of the gradient and going only a small step in that direction is that AI safety studies mostly consider practically unnoticeable changes of the inputs [6, 5]. Whereas the gradient of an arbitrary function could yield any value, the distortion should stay in reasonable bounds to mimic the behaviour of possible mismodellings or differences between data and simulation [5, 4].

The standard training procedure normally only needs gradients with respect to the parameters of the model itself (weights, bias terms) that are adjusted during backpropagation. The adversarial attack as described here only works if also the inputs carry a computation graph that allows differentiation. This has to be considered in the implementation [10] and adds some layer of complexity to be taken care of considering the memory usage.

Comparison with random noise A difference with respect to the previously described method of randomly adding noise is that the inputs are always shifted by exactly the same fixed amount ϵ (for the scaled version of the input distributions). The direction can vary per sample, but the absolute impact stays the same. This is different to the distortion with a noise term, where the absolute impact is not strictly predictable and only given by a probability density function, introducing randomness [6, 2]. Another way to implement the Gaussian noise aims at the construction of random distortions that are structurally similar to the FGSM attack, using a fixed stepsize [60]. With

$$x_{\text{Noise,fixed impact}} = x_{\text{raw}} + \epsilon \cdot \text{sgn}\left(\mathcal{N}\right), \tag{1.10}$$

it would be possible to compare the impact of both strategies also quantitatively, as the absolute magnitude would always be identical. Also theoretically, the robustness of neural networks has been studied. In a framework described in Ref. [65], the robustness to perturbations

obtained with random noise, compared to adversarial attacks, scales with $\sqrt{n}$, where n is the input dimension.

Theoretical limitations However, already when looking at the illustration of the FGSM attack, it becomes visible that the effect is not necessarily replicating a global worst-case scenario [64]. Depending on the actual properties of the loss surface, the adversarial attack could shift the inputs also into local minima (or at least harmless regions), if the limiting parameter is chosen unluckily. On average, with small distortions only, it is still expected that in a given region, the attack will maximally confuse the model. Other methods could construct the adversarial inputs with a different technique that takes care of the loss at the designated point for the new input and could decide to choose a different ϵ [64]. In general, the FGSM attack is only one example to construct adversarial inputs. It is possible to build the adversarial inputs themselves using a second neural network that works as an adversary to the originally concerned model [63, 4]. More limitations and possible extensions of the method will be discussed throughout the text, for example in Section 4.1.3 or Chapter 5.

1.6.3 Improving robustness with adversarial training

As has been pointed out, there are various techniques that can cause performance drops by applying adversarial attacks. To some extent, fortunately, improving the robustness of deep neural networks can be addressed with dedicated methods as well. The basic paradigm that always plays a decisive role is that there is a tradeoff between performance and robustness [63]. Therefore, models that do not reach peak performance, but show acceptable evaluation metrics might be preferred if instead they are more robust to mismodellings.

The approach that will be followed in this thesis is a simple type of adversarial training that injects perturbed inputs already during the training phase [63]. This differs from the usual way to only add the distortion to the test samples after the model has been trained on unperturbed training samples. In fact, when using adversarial training in the way it is done here (as depicted in Fig. 1.8), the neural network never sees the raw inputs during the whole training step [63, 66, 64]. The idea is that by applying the FGSM attack continuously to the training data (for every minibatch, i.e. with every intermediate state of the model after updating the model parameters), the network less likely learns the simulation-specific properties of the considered sample, but instead attempts to generalize better [2, 63].

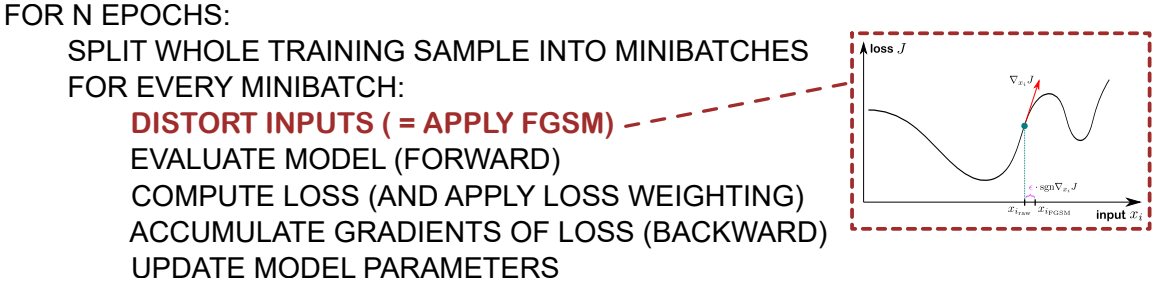

Figure 1.8: Conceptual design of an adversarial training algorithm. The inputs are distorted prior to the forward and backward passes, with the help of the FGSM attack. The standard training algorithm denoted in black is based on Ref. [52], the modified implementation for adversarial training is demonstrated in Ref. [10].

Madry et. al. [64] have shown that this is an effective method to reduce susceptibility to first-order adversaries, obtained from an FGSM attack. In that sense, adversarial training could

also be described as a regularization technique, but a more systematic one than only randomly smearing inputs (one example of data augmentation), randomly deleting connections (dropout), or assigning a probability to the different targets to be wrong (label smoothing) [2].

The principle behind this technique again involves the linearity of neural networks to which the high susceptibility is attributed. Adversarial training can be interpreted as a method that adjusts the loss surface to be locally constant around the inputs and that downsizes the impact of perturbations evaluated with a high-dimensional linear function [2]. Slightly distorted inputs then can not raise the loss function by large amounts, because it is almost flat in the vicinity of the raw inputs [62]. This can be seen as a geometrical problem that tries to flatten some manifold.

When evaluating this adversarially trained model with distorted test inputs (adding noise or again applying the FGSM attack), the model should be more robust to those modifications and the performance should not drop as much as with the generic training. The price for the increased robustness is of course that the maximally achievable performance on raw inputs will be reduced with respect to the basic training [2]. The goal will be to first of all show that this technique is applicable to heavy-flavour tagging and that finding a balance between susceptibility and performance is feasible. Possible implications when evaluating the two different models (basic / adversarial training) on data could help reduce concerns regarding mismodellings that might become visible when comparing data and simulation (and deriving scale factors) [4].

2 Classifier (DeepCSV) training

2.1 Architecture

2.1.1 Hard- and software, technical details

The implementation of the neural network and all related studies are done in Python and utilize the PyTorch [67] package. It offers the necessary data structures and operations to build neural networks and is highly customizable, making it a popular framework for machine learning tasks [52]. The core component is the class `Tensor`, which allows to compute gradients, but at the same time offers a behaviour similar to NumPy arrays [68] when it comes to efficient manipulations of several elements at once. Just like a model itself, tensors with their computation graph can be transferred to a Graphics Processing Unit (GPU). The available nodes of the SLURM batch system at the RWTH Compute Cluster offer access to a GPU of type NVIDIA Tesla V100 [69], which fascilitates parallel computations used to update the trainable parameters of the model. More aspects of the technical details that are relevant for the specific use case are discussed in Section 2.4.1, as the model and inputs first need to be introduced.

2.1.2 Model

General setup

For all subsequent studies, it is important to have full control over a neural network that approaches the classification task similar to a widely used tagger like DeepCSV, as described in Ref. [8]. Replicating a commonly used architecture is desirable because the outputs and derived discriminators are a key ingredient to physics analyses that rely on heavy-flavour tagging. One example publication that explicitly uses DeepCSV to search for a Higgs boson decaying to charm quarks in the resolved-jet topology is described in Ref. [50]. Hence, it is beneficial to use a model that shares most of the properties of the popular tagger, including the network's structure and resulting performance, to draw conclusions that could be generalized to an actual application. The following description is based on Ref. [8]. In general, the type of the neural network is a fully-connected sequential model of dense layers. It takes 67 inputs and provides one node for each of those in the input layer. The chosen variables are either kinematic quantities or offer discriminating power between the different flavours in any way. This is one more variable than what is mentioned in Ref. [8] and reflects the updated definitions as found in the input files. A more detailed explanation of how the input variables are used and what their physical interpretation is follows in Section 2.2 (especially Section 2.2.2).

Five hidden layers with 100 nodes each are consecutively placed after the input layer and the network yields four outputs. From the output nodes, the probabilities for a jet to belong to one of the categories b, bb, c or udsg (light) can be read off. The model is visualized in Fig. 2.1.

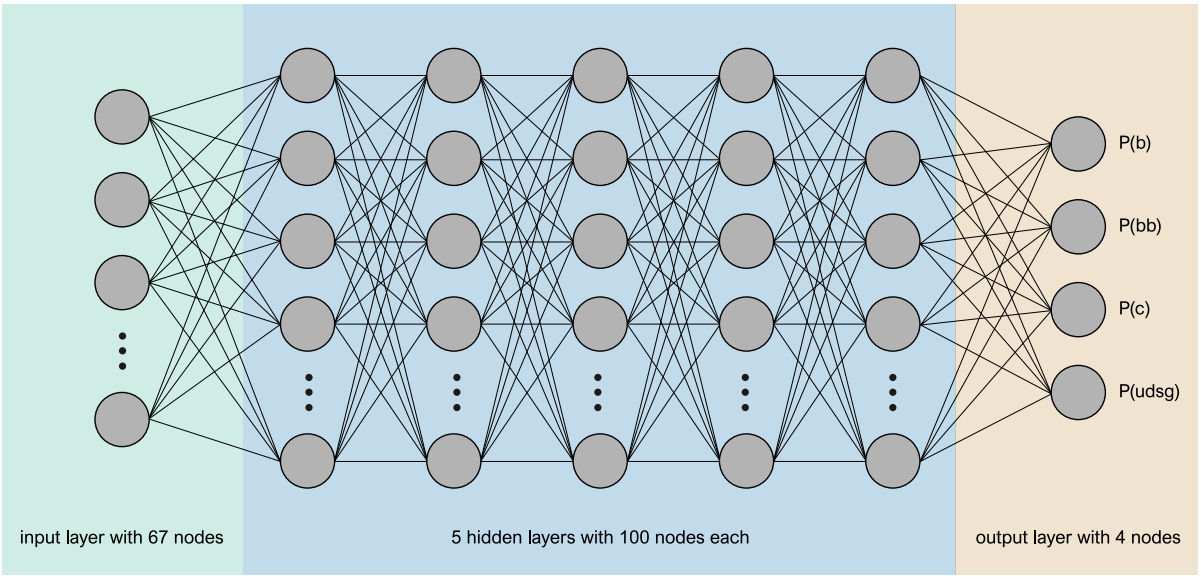

Figure 2.1: Visualization of the network architecture. Modified adaptation of Fig. 1 in Ref. [9].

In the setup described above, the number of trainable parameters [52] that represents a simple metric for the complexity of the model can be obtained as follows:

$$\begin{aligned}
\text{trainable parameters} &= (\text{weights}) + (\text{bias terms}) \\
&= (\text{input nodes}) \cdot (\text{nodes in hidden layer}) \\
&\quad + (\text{hidden layers} - 1) \cdot (\text{nodes in hidden layer}) \cdot (\text{nodes in hidden layer}) \\
&\quad + (\text{nodes in hidden layer}) \cdot (\text{output nodes}) \\
&\quad + (\text{hidden layers}) \cdot (\text{nodes in hidden layer}) \\
&\quad + (\text{output nodes}) \\
&= 67 \cdot 100 + 4 \cdot 100 \cdot 100 + 100 \cdot 4 + 5 \cdot 100 + 4 \\
&= 47604
\end{aligned}$$

and it has been cross-checked that the number of parameters of the model that require gradients equals that number. The first three terms in the calculation count all the connections between nodes (weights), while the last two rows add one bias term per node for the hidden layers and for the output layer. During the training, those parameters can be updated by backpropagation.

Activation functions

This implementation (see Ref. [8]) utilizes the Rectified Linear Unit (ReLU) activation function for all hidden layers, as recommended for the majority of feedforward neural networks [52, 2]. It is defined as

$$f_{\text{ReLU}}(z_i) = \max(0, z_i), \tag{2.1}$$

where

$$z_i = \left(W \cdot \vec{x} + \vec{b}\right)_i \tag{2.2}$$

is the (linear) pre-activation of a single neuron [52, 2].

For the output layer, the activation is computed with the Softmax function, which is performed frequently for multi-class classification problems [2]:

$$f_{\text{Softmax}}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}, \tag{2.3}$$

where z_i is defined as above. The idea behind choosing the Softmax activation is that the resulting output nodes carry values that are bound between 0 and 1 and that they are normalized, summing up to 1 [2]. This allows interpreting the output nodes directly as probabilities, more specifically, the tagger output reports with which probability a given input (jet) belongs to which class (flavour) [8]. The computation happens element-wise, with i being a placeholder for the four different output categories.

Loss function

As explained in the introduction (see Section 1.5), the training algorithm requires a metric that measures the discrepancy between the predictions and the true labels, technically by comparing two probability distributions [52].

If one would differentiate between two categories only, the widely used binary cross entropy loss function would be sufficient to separate, for example, a signal ($y = 1$) from a background process ($y = 0$) [70]. Then, the binary cross entropy with predicted signal probability x reads

$$L(x, y) = -\left(y \log(x) + (1 - y) \log(1 - x)\right) = \begin{cases} -\log(x), & \text{for } y = 1; \\ -\log(1 - x), & \text{for } y = 0. \end{cases} \tag{2.4}$$

For the classification task studied here, there are four categories, and so, the binary cross entropy function needs to be extended to a categorical cross entropy function [52]. Using that x is now a tensor that carries the predictions for all M target classes for an individual sample, this loss can be specified as (see [71]):

$$\text{loss}(x, \text{class}) = -\log\left(\frac{e^{x[\text{class}]}}{\sum_{j=0}^{M-1} e^{x[j]}}\right) = -x[\text{class}] + \log\left(\sum_{j=0}^{M-1} e^{x[j]}\right). \tag{2.5}$$

This combines a negative log likelihood with a softmax function (the logarithm improves numerical stability when dealing with small numbers), where likelihood refers to the probability associated with the correct class [52]. The loss function complies with the requirement to be high when the likelihood is low, and vice-versa [52]. While training the model, the cost function is averaged over the current (mini-)batch, which reduces the dimension from $\texttt{length}(\texttt{batch}) = N$ to a scalar, allowing the subsequent computation of the gradient [71]:

$$\text{loss}(\texttt{batch}) = \frac{1}{N} \sum_{i=0}^{N-1} \text{loss}(\texttt{batch\_predictions}[i], \texttt{batch\_classes}[i]). \tag{2.6}$$

In a more general form, independent of the implementation described above, the categorical cross entropy (CCE) loss [2, 72] can be written as

$$L_{\text{CCE}}(\vec{x}, \vec{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{j,i} \log\left(\hat{y}_{j,i}\right), \tag{2.7}$$

where M counts the number of classes, N is the batch size, $y_{j,i}$ is the true output, which yields 1 if the i-th jet is part of class j (and 0 otherwise) and where $\hat{y}_{j,i} = y_j\left(x_i | w, b\right)$ denotes the

predicted value of class j for the i-th jet when utilizing all weights w and bias terms b of the model.

It is possible to train the network with the function described in Eq. (2.7), but there are caveats related to the class imbalance and the dependence on jet kinematics that involve a more advanced treatment. Two reweighting techniques are introduced in Section 2.3.

Dropout

As a simple regularization technique, dropout layers [73] are used to prevent overfitting. In all hidden layers, with a probability of $10\,\%$, the activation of each node is set to 0. These randomly chosen nodes therefore do not contribute during forward calls while training the network and are only active while evaluating the model on validation or test inputs [74]. This method can also be interpreted as averaging the resulting outputs over an ensemble of different neural networks [75], which would otherwise be unfeasible due to time and memory constraints for the training [2].

Optimizer and learning rate

The Adaptive Moments Estimation (Adam) optimizer controls the weight updates during the training with an adaptive learning rate for each model parameter while iterating over the minibatches [76]. An adaptive learning rate is preferred over a constant or purely decaying learning rate, because the training might oscillate around or between local minima, instead of reaching them eventually [77]. In general, too high learning rates might prevent the model from converging, but taking too small step sizes can slow down the training and does not necessarily result in a global minimum [2]. Updating the model parameter θ_t with the next minibatch $t+1$ is done by evaluating

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \tag{2.8}$$

where the mean (first moment) $\hat{m}_t$ and variance (second moment) $\hat{v}_t$ of the gradients of the loss function with respect to the model parameters are taken into account [77]. θ_t represents a weight or bias term after the previous and t-th iteration (minibatch). In this context, η and ϵ are tuneable, along with two additional hyperparameters β_1 and β_2 used to calculate the moments (defaults in PyTorch are: $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$ [78]; for η, the default of 0.001 is replaced, see below).

Additionally, to avoid that the training "overshoots" while approaching the minimum of the loss surface, the parameter `lr` of the optimizer (also called η in Eq. (2.8) above) is updated for every epoch based on the following rule:

$$\texttt{lr}(epoch) = \frac{\texttt{lr}(epoch = 0)}{1 + \frac{epoch}{30}} \qquad \text{with} \qquad \texttt{lr}(epoch = 0) = 0.0001. \tag{2.9}$$

This is effectively a learning rate schedule [79], with a decaying learning rate that is initialized with 0.0001 at the beginning of the training, interfaced with the usage of adaptive moments per minibatch in each epoch.

2.2 Preparation of the data sets

2.2.1 Used files and processes

Different types of input files are used to train a classifier that is not limited to one specific process. This is to make sure the tagger can in principle generalize to various use cases when

applied in an actual physics analysis. Instead of learning only some distinct properties of for example a $t\bar{t}$ process, a mixture of $t\bar{t}$ and QCD multijet samples will be used [8]. More specifically, there will be files that contain semileptonic $t\bar{t}$ events, where a large fraction of b jets is expected from the weak decay of top quarks [15]. The QCD multijet samples from simulated proton-proton collisions complement the chosen training samples with a large fraction of udsg jets (see Sections 1.4.2 and 2.2.3). To model the physics processes, the event generator MADGRAPH5 [80] is used for the QCD samples, whereas POWHEG 2 [81] simulates the top quark pair production. In both cases, PYTHIA 8 [16] subsequently handles the parton showering and hadronization. The detector response is simulated with GEANT4 [82] to take the interactions between particles and material into account [8].

This combined selection allows to cover an extensive range of the phase space in jet-p_{T} and jet-η relevant for studies that concern processes within (or beyond) the standard model [83]. Table 2.1 lists the names of the chosen input files together with the number of events and jets before any cleaning or preprocessing has been applied.

Table 2.1: Used processes with their respective filenames. The event count and the number of jets that could be retrieved from the chosen samples before any cleaning are shown in the additional columns.

Sample name	Events	Jets
`QCD_HT300to500_TuneCP5_13TeV-madgraph-pythia8`	60316577	428503277
`QCD_HT500to700_TuneCP5_13TeV-madgraph-pythia8`	56207744	430806556
`QCD_HT700to1000_TuneCP5_13TeV-madgraph-pythia8`	19761895	156090013
`QCD_HT1000to1500_TuneCP5_13TeV-madgraph-pythia8`	16595628	135238130
`QCD_HT1500to2000_TuneCP5_13TeV-madgraph-pythia8`	11634434	97728944
`QCD_HT2000toInf_TuneCP5_13TeV-madgraph-pythia8`	5941306	50313754
`TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8`	43732445	382423670

Several thousand "small" files are accumulated in packages of 50 elements into "larger" files for storage purposes (using the resulting 278 files instead of 13853 files will facilitate the preprocessing and splitting) and the branches have been flattened to provide the information per jet, not per event[1].

Additional to this setup with a mixture of processes, a dedicated reweighting reduces learning of undesirable behaviour related to flavour-dependent kinematic variables like the transverse momentum of the jet [8] and is further described in Section 2.3.

2.2.2 Description of input variables

The input layer takes high-level variables [8, 11] and consists of 67 nodes, which are related to the observables described in the following (see Tables 2.2,2.3 and 2.4 and Ref. [8]). The different types of inputs that can be classified based on the physics objects they are related to or for how many constituents of the same type they are available. More specifically, taking the physical meaning into account, there exist global (jet) variables that are given once per

[1]This is implemented with the help of the python-packages uproot4 [84] and awkward1 [85]. Information of the TTree named `Events` of the `.root`-files (obtained with the `PFNano` framework developed to test jet algorithms [86, 87]) is extracted and later on converted to NumPy [68] arrays for further usage. The `keys` under which the variables can be accessed inside the original files are additionally quoted in parantheses.

jet, then there are track variables that are stored for several tracks inside the jet and the third group contains information about one secondary vertex associated with the jet (see Sections 1.3.2 and 1.4 for an introduction of the reconstruction of tracks and secondary vertices). From a technical point of view, all variables are stored as separate columns per jet. In all cases where a significance is specified, this refers to the value of an observable devided by its uncertainty.

Table 2.2: Global jet variables. 13 properties that are stored on a per-jet basis have been selected. The definition of "jet" variable includes those features that are related to tracks, but are only available once, not for several tracks due to their global meaning. Adapted from Ref. [8].

Short name	Description
Jet η	The first, purely kinematic variable stores the pseudorapidity of the jet. (`Jet_eta`)
Jet p_{T}	Also, the transverse momentum of the jet is considered as an input variable. (`Jet_pt`)
Track Jet p_{T}	This is the transverse momentum of the jet, obtained from the sum of the transverse momenta of the tracks only. (`Jet_DeepCSV_trackJetPt`)
Track SIP 2D (3D) Val (Sig) Above Charm	In general, the impact parameter (IP) vector is the vector pointing from the primary vertex to the point of closest approach of the track. The signed impact parameter (SIP) can be obtained by taking the modulus of this quantity, but keeping the information about whether the angle between the IP vector and the jet axis is smaller than $\pi/2$ (SIP > 0) or not (SIP < 0). This can be done in three dimensions, in two dimensions (transverse to the beam line), or along the beam line in one dimension. Then, for a given track, the value as well as the significance (SIP$/\sigma_{\mathrm{SIP}}$) is stored. For the four variables considered here, the two-dimensional (2D) and three-dimensional (3D) SIP value and significance are taken for the first track that raises the combined invariant mass above the threshold of the charm quark mass ($1.5\,\mathrm{GeV}$). (`Jet_DeepCSV_trackSip2dValAboveCharm`, `Jet_DeepCSV_trackSip2dSigAboveCharm`, `Jet_DeepCSV_trackSip3dValAboveCharm`, `Jet_DeepCSV_trackSip3dSigAboveCharm`)
Track Sum Jet ΔR	First, the four-momentum vectors of the tracks are summed to get a total four-momentum. Then, the angular distance between this summed four-momentum and the jet axis is computed, using $\Delta R = \sqrt{(\Delta\phi)^2 + (\Delta\eta)^2}$. (`Jet_DeepCSV_trackSumJetDeltaR`)
Track Sum Jet E_{T} Ratio	For this variable, the transverse energy of the total summed four-momentum of the tracks is divided by the transverse energy of the jet. (`Jet_DeepCSV_trackSumJetEtRatio`)
Vertex Category	With the vertex category, three cases can be defined that describe the result of the secondary vertex reconstruction algorithm. (`Jet_DeepCSV_vertexCategory`) • RecoVertex: at least one secondary vertex has been reconstructed in the jet. • PseudoVertex: the reconstruction has not found a secondary vertex and no fit is performed, but there are at least two tracks with a 2D impact parameter significance greater than two, whose combined invariant mass is $50\,\mathrm{MeV}$ away from the mass of the $\mathrm{K}^0_{\mathrm{S}}$. • NoVertex: this label is given to all jets that do not belong to one of the previous categories.
Jet N Secondary Vertices	This quantity lists the number of secondary vertices in the jet (of the RecoVertex category). (`Jet_DeepCSV_jetNSecondaryVertices`)
Jet N Selected Tracks	The number of selected tracks in the jet counts the number of tracks that remain after introducing special criteria related to the quality of the track reconstruction, criteria to reduce contamination from (for example) $\mathrm{K}^0_{\mathrm{S}}$ or Λ hadrons, or to minimize the contribution from pileup. The imposed conditions involve properties like a minimum p_{T}, a required number of pixel hits, a maximum threshold for the χ^2 of the track fit, or utilize the impact parameter or the angular distance between the track and the jet axis. (`Jet_DeepCSV_jetNSelectedTracks`)
Jet N Tracks η_{rel}	This is the number of tracks for which the property η_{rel} is available (has been computed out of the selected tracks). (`Jet_DeepCSV_jetNTracksEtaRel`)

Table 2.3: Track variables. The tracks are ordered by the significance of their 2D signed impact parameter (as defined in Section 1.4.2 and Table 2.2). Then, the six highest ranked tracks (the first one having the highest 2D SIP significance) are selected and further described by seven properties. Another eighth property (Track η_{rel}) is only included for the first four tracks of that ranking. Adapted from Ref. [8].

Short name	Description
Track Decay Len Val	This is the distance between the primary vertex and the track, at the point of closest approach between the track and the jet axis. In the data set, this feature appears for the first six tracks. (`Jet_DeepCSV_trackDecayLenVal_0` to `5`)
Track ΔR	For this input variable, the angular distance $\Delta R = \sqrt{(\Delta\phi)^2 + (\Delta\eta)^2}$ between the track and the jet axis is stored (six times). (`Jet_DeepCSV_trackDeltaR_0` to `5`)
Track Jet Dist Val	To obtain the following quantity (for the first six tracks), the distance between the track and the jet axis at their point of closest approach has been calculated. (`Jet_DeepCSV_trackJetDistVal_0` to `5`)
Track $p_{\mathrm{T,rel}}$	The next feature constitutes the track momentum, perpendicular to the jet axis (in other words: the track p_{T} relative to the jet axis), available for the first six tracks. (`Jet_DeepCSV_trackPtRel_0` to `5`)
Track $p_{\mathrm{T,rel}}$ Ratio	Based on the previous quantity, this feature now uses the track's $p_{\mathrm{T,rel}}$ and divides this value by the magnitude of the track momentum vector, again for the first six tracks. (`Jet_DeepCSV_trackPtRatio_0` to `5`)
Track SIP 2D (3D) Sig	In the way it is used here, only the significance of the signed impact parameter (2D or 3D) is saved for the first six tracks (ordered by 2D SIP significance). (`Jet_DeepCSV_trackSip2dSig_0` to `5`, `Jet_DeepCSV_trackSip3dSig_0` to `5`)
Track η_{rel}	This variable is defined as the pseudorapidity of the track relative to the jet axis and is stored for maximally four tracks, ranked by 2D SIP significance as described above. (`Jet_DeepCSV_trackEtaRel_0` to `3`)

Table 2.4: Secondary vertex variables. Additionally, there are eight input variables related to the secondary vertex (SV) with the smallest uncertainty on its (2D) flight distance (if such a secondary vertex exists, i.e. if a secondary vertex could be reconstructed $\rightarrow$ vertex category RecoVertex). The more specific definition of the chosen secondary vertex is omitted from now on, but always meant implicitly. Adapted from Ref. [8].

Short name	Description
Flight Distance 2D (3D) Val (Sig)	With these quantities, the 2D (or 3D) distances between the primary and secondary vertex are taken into account. The 2D (or 3D) significances are used in addition to the values. (`Jet_DeepCSV_flightDistance2dVal`, `Jet_DeepCSV_flightDistance2dSig`, `Jet_DeepCSV_flightDistance3dVal`, `Jet_DeepCSV_flightDistance3dSig`)
Vertex Energy Ratio	This is the energy of the secondary vertex, divided by the energy of the total summed four-momentum vector of the selected tracks. (`Jet_DeepCSV_vertexEnergyRatio`)
Vertex Jet ΔR	Here, the ΔR between the flight direction of the secondary vertex (or the total summed four-momentum of the selected tracks) and the jet axis is stored for vertex category RecoVertex (PseudoVertex). (`Jet_DeepCSV_vertexJetDeltaR`)
Vertex Mass	This is the corrected mass of the secondary vertex for jets in the RecoVertex category, which takes care of the observed difference between the flight direction and the momentum of the SV when not all particle were correctly reconstructed or associated with the SV. The correction uses $\sqrt{M_{\mathrm{SV}}^2 + p^2 \sin^2\theta} + p\sin\theta$, with the invariant mass of the tracks associated with the SV, M_{SV}, the SV momentum p and the angle θ between the SV momentum and the vector pointing from the PV to the SV [8]. As the mass of the SV is directly related to the mass of the decaying hadron, this quantity offers high discriminating power. For jets in PseudoVertex category, this variable lists the invariant mass of the total summed four-momentum vector of the selected tracks. (`Jet_DeepCSV_vertexMass`)
Vertex N Tracks	For vertex category RecoVertex, this property counts the number of tracks associated with the secondary vertex (no. of selected tracks for cat. PseudoVertex). (`Jet_DeepCSV_vertexNTracks`)

2.2.3 Definition of the targets

To get the truth labels (targets) for the jets in the data set, the hadron-based jet flavour definition as described in Ref. [88] is used. In the dataformats `MiniAOD` [89] and `NanoAOD` [87], the required hadron-level information is stored independent of the event generator, whereas the availability of detailed information per parton depends on the generator and does not necessarily match the other definition unambiguously [88].

All categories should be mutually exclusive and, where applicable, allow for a versatile usage (e.g. splitting into subcategories allows a more specific discrimination between flavours which can give a hint on the production process) [8]. Adding probabilities afterwards to include more than one category or taking ratios of probabilites to get a combined discriminator is always possible.

Four distinct categories have been chosen based on the following conditions, as described in Refs. [88] and [8]:

- `Jet_hadronFlavour == 5`
  At least one b hadron[2]is clustered inside the jet. This can be further specified by counting the number of b hadrons.
  - `Jet_nBHadrons <= 1` (b jets)
    Less than two b hadrons have been clustered in the jet (together with the already imposed condition on the hadron flavour, this is exactly one b hadron).
  - `Jet_nBHadrons > 1` (bb jets)
    More than one b hadron has been found after the jet clustering took place. These likely originate from gluon splitting, but their properties are mainly defined by the two (heavy-flavour) b hadrons, which explains the assigment to this main category.

  For the cases where differentiating between the number of b hadrons is not necessary, both subcategories can be combined to include b and bb jets (sometimes abbreviated with a capital B in various figures showing the performance of the classifier).
- `Jet_hadronFlavour == 4` (c jets)
  No b hadrons, but at least one c hadron results from the jet clustering.
- `Jet_hadronFlavour != 5 && Jet_hadronFlavour != 4` (udsg/light/l jets)
  Neither b, nor c hadrons are present in the jet. Thus, these jets are defined as udsg jets, including light-flavour, quark-initiated jets and those originating from gluons. Due to the small mass of uds quarks (see Fig. 1.1), and with gluons being massless, when discriminating the light jets from heavy-flavour jets, the properties of the aforementioned light jets are similar such that all four hypothetical subcategories (u, d, s, g jets) can be combined into the udsg category.

The usage of the parton flavour would be necessary to split the different types of udsg-flavour jets into the respective quarks u, d and s or into gluon initiated jets. DeepCSV however does not do this, and only the granularity of the c hadron category is increased in certain versions of the classifier in a similar way as described for the b hadrons, splitting the c and cc category [8]. The successor, DeepJet, splits the udsg-category only one step further into quark- or gluon-initiated jets [51]. For the purposes of this study, the four categories as described above are sufficient and already allow comparisons with commonly used versions of jet taggers in CMS.

[2]More precisely, this definition uses "ghost" hadrons [90, 8, 88]. In a process called "ghost-matching" [48] or "ghost association" [8], the four-momenta of the immediately decaying hadrons are rescaled, usually by the factor 10^{-18} [88]; then these soft "ghosts" are added to the list of particles (keeping the direction of the four-momenta), with which a reclustering is performed [88, 48]. This results in an almost identical jet collection, as the rescaled, soft "ghosts" contribute only marginally [8, 88]. For simplicity and because the details related to jet clustering and pileup subtraction [90, 43, 44] are beyond the scope of this thesis, the term "ghost" will be omitted from now on.

2.2.4 Cleaning procedure

The samples in their original form might include variables that are filled with unphysical values or could be part of the phase space that is not covered by the tracker [8]. Two methods will be used during a cleaning step: dropping whole jets based on certain conditions or keeping the jet, but choosing a special default value for the variables that offer no or faulty information.

Conditions to delete whole jets

As described in Ref. [8], there are constraints for the kinematic variables of the jets to stay within the tracker acceptance with sufficient reconstruction efficiency and to achieve a good transverse impact parameter resolution [8, 48]. Therefore, only jets with a pseudorapidity of $|\eta| < 2.5$ are considered and another preselection limits the transverse momentum of the jet to $20\,\mathrm{GeV} < p_\mathrm{T} < 1\,\mathrm{TeV}$, other jets are discarded directly [8].

Additionally, the angular distance between the summed four-momentum of the tracks and the jet axis is required to be less than 0.3. The outliers of the $\Delta R\,$(summed tracks, jet) variable would otherwise exceed additional selection requirements after the application of the clustering algorithm [8].

Handling of defaulted values

There are different problematic conditions where choosing a default value is unavoidable. In some cases, the jets already include variables that are filled with `-Inf`-, `+Inf`- or `NaN`-entries or have been given a preliminary default value of -999 or -1. Then, there are cases where the produced samples show randomly assigned large numbers (examples are numerical values like $\pm 10^{34}$). A proper way to deal with the various different special cases has to be found that covers all visible eventualities in a similar way. Prior to explaining the strategy to assign the defaults, the following paragraphs explain why doing so is necessary.

The reason for the occurrence of default values is mainly related to the case where a secondary vertex could not be reconstructed [8]. Variables that depend on (at least one) secondary vertex are then set to values like `NaN` or -999. Such problems appear mostly for global input variables related to the jet or one designated secondary vertex. Another group of default values results from the variables that can only exist if the threshold of the charm quark mass is reached, namely the track signed impact parameter 2D (3D) value and significance for the first track that raises the invariant mass of all tracks above the charm quark mass. If the invariant mass of the tracks does not exceed $1.5\,\mathrm{GeV}$ (related to the charm quark mass), the aforementioned variables carry the default value -1 [8].

By investigating the distributions, it becomes evident that another type of default values has to be imposed manually for the track variables to eliminate the enormously large values. Most track properties are given for six tracks, while the pseudorapidity of the track relative to the jet axis is only available for four tracks. In any case, independent of how these tracks are ordered, a necessary condition to have meaningful values is that there exist enough tracks in the jet for which those variables have been reconstructed (for the definitions, see Tables 2.2, 2.3 and 2.4 and Ref. [8]). For example, the first track ([0] due to zero-indexing) only shows a meaningful variable for the pseudorapidity between the track and the jet axis if there is at least one track for which the computation of that quantity has succeeded. A similar condition is imposed for the second, third and fourth track. Correspondingly, the first (second, ..., sixth) track needs a default value for all remaining per-track variables if the number of selected tracks in the jet is less than $1(2, \ldots, 6)$.

As a next step, default values need to be defined, which shall be used as a placeholder where meaningful values are not available. A short review of the chosen method to assign the defaults and possible alternatives is given below. In the current setup, this placeholder is assigned consistently per variable and is chosen such that it is located outside of the natively filled range of the distribution, but close to it [91]. The actual position has been identified by calculating the minimum of the distribution for a few example files, taking the average value of those minima and subtracting a small number (here: 0.001) [92]. It has been found that the closer the default values are located to the bulk distribution, the faster the convergence of the model [92]. Placing defaults in other regions that are not filled could otherwise introduce large discontinuities when evaluating the loss function, vanishing or exploding gradients, ultimately leading to a reduced performance [52, 2, 93].

Another possibility to assign the default values is to put them at zero [8] (after the scaling, which will be described in Section 2.2.5). This would have the benefit that default values do not contribute when updating the model parameters (multiplying zero with any weight will still result in a zero, practically erasing the influence of that input variable during back-propagation). The problem with putting default values at zero on the other hand is that there could exist true zero values that are actually already part of the generic distribution. Then, there would be no difference between "true" and "false" input variables although they are fundamentally different and should not interfer. Several other options to handle missing data exist, deciding per variable whether placing defaults at zero is feasible, or if putting them at the lower or upper end of the distribution is safer. Procedures like K-nearest neighbours [93] can be used to identify K samples in the training data that are complete (no defaults) and have a minimal distance to the sample in question that lacks information for certain input features [93, 2]. Averaging the predictors from those K closest samples would give an appropriate estimate to replace the default value [93]. However, this approach does not incorporate constraints based on physics. Technically, one could also build different classifiers for different categories of jets that differ by their number of default values or the vertex category. But then, these results would need to be combined, not all possible correlations between inputs would play a role during training, and the sample sizes per individual training would be much reduced [8, 94].

The fractions of jets that carry a default value for the individual input variables are displayed in Figs. 2.2, 2.3 and 2.4, for global jet variables, track variables and secondary vertex variables. In general, the global variables are less likely filled with defaults, as they do not depend too much on the number of tracks or the vertex category. The secondary vertex variables practically only offer real information if the vertex category variable is not set to NoVertex (i.e. `vertexCategory` $\neq 2$). A large fraction of defaults occurs for the track variables, as these depend on the number of (selected) tracks in the jet. The abundance of defaults counter-intuitively increases with the number of tracks (for all four flavours individually and for the inclusive selection in the fifth row, see Fig. 2.3). This is expected due to the construction of the defaults for those variables (e.g. the sixth track variable can only be filled if there is a sixth track, which is less likely than having five selected and fully reconstructed tracks in the jet).

There is a striking difference between the different flavours related to the fraction of defaults: particularly the jets in category bb have much less defaults than for example the light jets. This is due to the higher track multiplicity related to the decay of two b hadrons that are part of the same jet [8, 47]. As will be discussed in Section 2.2.5, the overall fraction of bb jets is the lowest out of all considered categories in the prepared data set. On the other hand, those jets offer the highest fraction of meaningful input values. It will therefore be important to keep an eye on a balanced reweighting that emphasizes the importance of hard or easy to classify samples. Some techniques that involve the flavour distribution as well as their inherent difficulty levels are explained in Section 2.3.

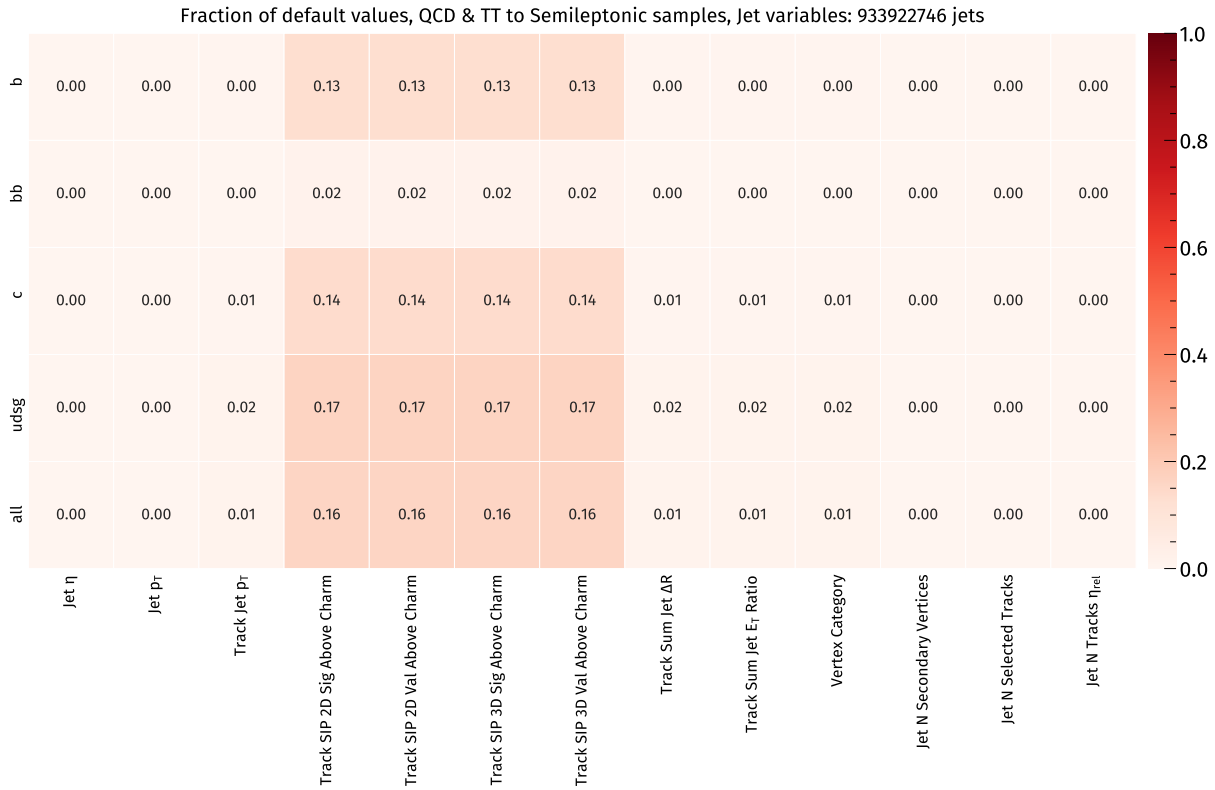

Figure 2.2: Fraction of default values for global jet variables, split by flavour or obtained from all available jets.

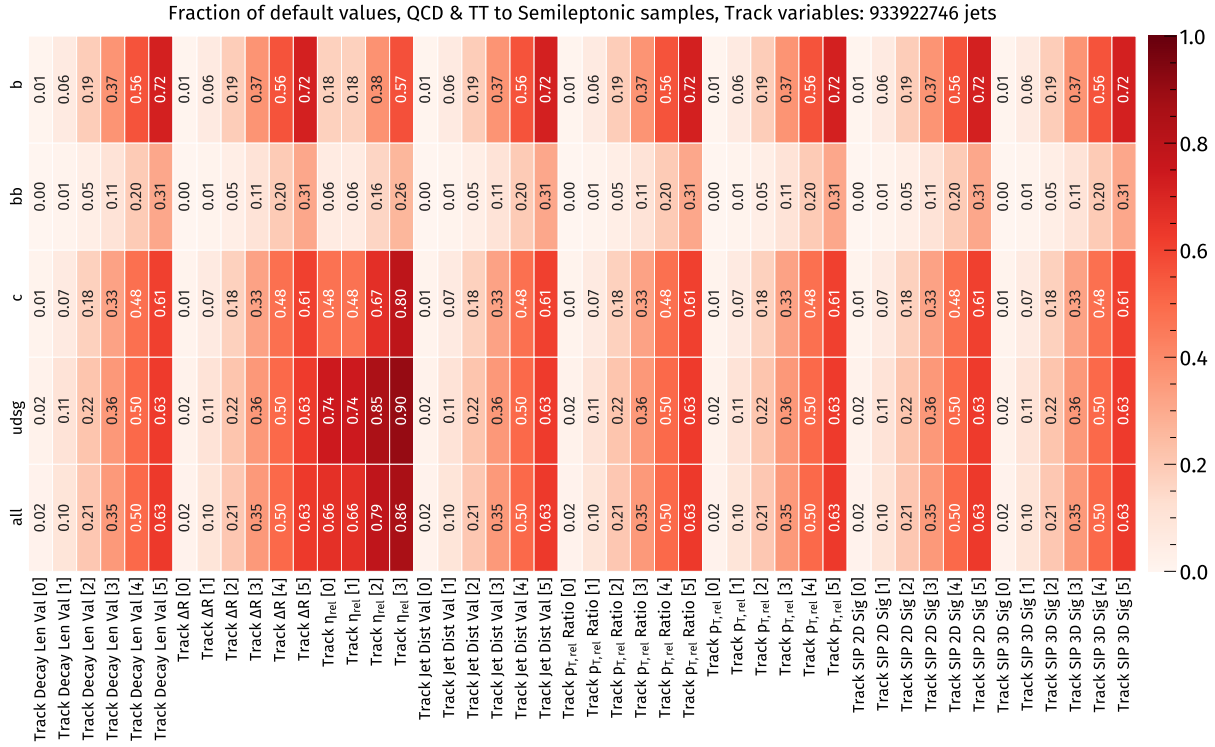

Figure 2.3: Fraction of default values for track variables, split by flavour or obtained from all available jets.

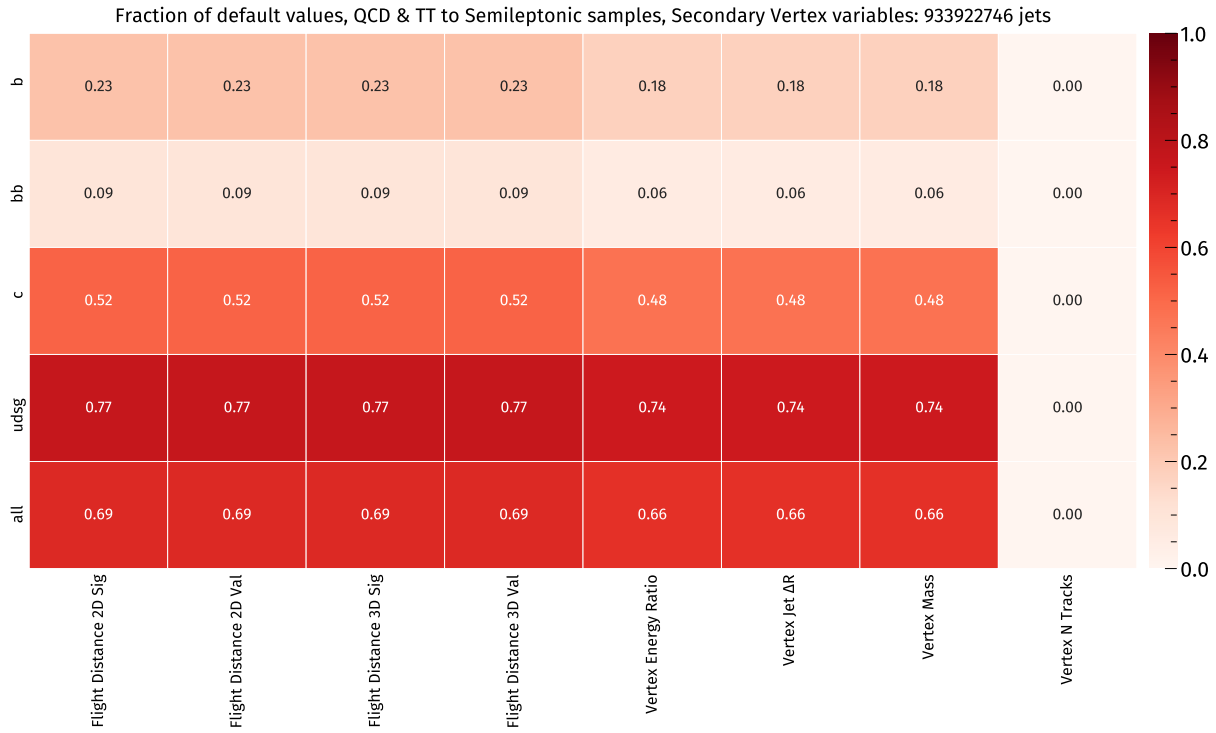

Figure 2.4: Fraction of default values for secondary vertex variables, split by flavour or obtained from all available jets.

2.2.5 Further preprocessing

So far, the preparations have resulted in cleaned data sets, available as NumPy arrays. The next steps will utilize these intermediate results in an additional preprocessing task and store them ready for usage as PyTorch tensors.

Splitting the whole sample

Before the inputs can be used in the training, all available jets are randomly separated into three mutually exclusive sets [52, 93]. When evaluating the model, it is necessary to provide a sample of jets on which the network has not been trained on. Otherwise the resulting performance metrics could be biased as the model might just have explicitly memorised the mapping of inputs to outputs instead of learning their interdependence, correlations or more complex structures [2]. This explains the first split of the whole sample where $20\,\%$ of the available jets are put into a special test set. During the training, one wants to make sure that the model is able to generalize [2]. Therefore, a simple approach to check how the model deals with new (unseen) data is to evaluate the loss function on a separate set of inputs after every training epoch. This is the reason for another division of the previously obtained preliminary training set into the actual training samples ($90\,\%$ of $80\,\% \hookrightarrow 72\,\%$ of the total) and those used for validation ($10\,\%$ of $80\,\% \hookrightarrow 8\,\%$ of the total)[3]. The monitoring of training and validation loss helps to identify possible overfitting. It also allows the usage of early stopping or other techniques like an adaptive learning rate that depends on the behaviour of the training and validation loss [2]. The basis for these modifications is already implemented, but for this study, the validation loss is only interpreted qualitatively and just serves as a guideline as discussed in Section 2.4.3.

[3]The implementation is done with the `sklearn.model_selection.train_test_split()` function of scikit-learn [95].

The distribution of samples and flavours after the available jets are split into the different subsets is given in Table 2.5, with the cleaning of Section 2.2.4 already applied.

Table 2.5: Distribution of samples into training, validation or test set and the respective flavour content.

Sample → / Flavour ↓	Training	Validation	Test	All samples
b	65171355	7241434	18096125	90508914 (9.7 %)
bb	4849162	538355	1348007	6735524 (0.7 %)
c	54557491	6065387	15154480	75777358 (8.1 %)
l	547846145	60868752	152186053	760900950 (81.5 %)
All flavours	672424153 (72 %)	74713928 (8 %)	186784665 (20 %)	933922746 (100 %)

As can be seen, there is a large class imbalance between the different flavours, with most data stemming from category l (udsg jets). There is no specified target distribution of flavours (like it has been done in Ref. [8]); instead, simply all available jets from simulated QCD multijet and semileptonic $t\bar{t}$ processes that remain after the cleaning are considered. This requires further treatment that will be described in Section 2.3, where the distribution of flavours plays a major role during reweighting.

Scaling the input distributions

The range of the different input variables varies, some are already located close to zero, others could have numerical values up to 1000, some have positive integer values only, and others could yield any real number. It has been found that in such cases, a scaling of the input variables can speed up the training and can prevent that some input variables contribute more than others by the pure fact that their numerical values are larger [8, 52, 2].

The new input shapes will be centered at 0 and have a standard deviation of 1. In practice, the normalization is done by computing the mean value μ_{original} of the input distribution, as well as the standard deviation σ_{original}. These values are used to shift and scale the input shapes using the following formula [2]:

$$x_{\text{normalized}} = \frac{x_{\text{original}} - \mu_{\text{original}}}{\sigma_{\text{original}}}. \tag{2.10}$$

The mean value and standard deviation are derived only from the "bulk distribution", excluding the default values [93]. Just the training samples are taken into account to calculate the normalized (or, used synonymously, scaled) quantities. This is done using all available training input files at once [52], which is necessary because the inputs are spread over more than one file (here: 278 files) with slightly varying distributions, as visualized in Fig. 2.5. If one would compute the mean and standard deviation individually for each sample, an input with a given value could end up in different locations just by the fact that it was part of, say, sample A instead of sample B, which differ by their respective mean values, for example. As the files are merged together (or in other words, during training the batches are drawn from all available files), this would wash out the discriminating power of that input variable.

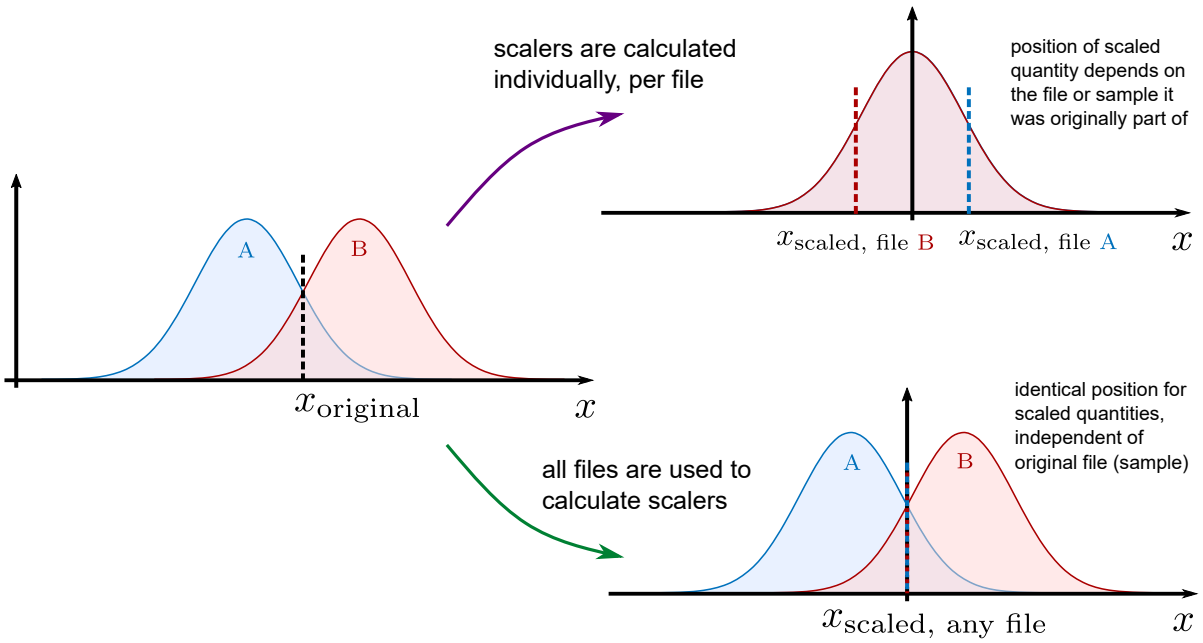

Figure 2.5: Visualization of two different approaches for the scaling of the input distributions. The first method calculates individual scalers per file, while the second takes all samples from all training files into account. Gaussian distribution adapted from Ref. [61].

2.3 Reweighting

2.3.1 Calculating weights

One property of the chosen samples for the training phase is the class imbalance between the four different categories, namely b, bb, c and udsg (light) jets. This is important, because the model should perform well for all classes and inclusive metrics like accuracy are misleading in such cases [96]. In particular, the majority class (udsg) would drive a high accuracy, while the performance on other flavours could be significantly worse. Another reason to take a closer look at the class imbalance and possible strategies to mitigate this effect are the tagger outputs and discriminator shapes going to be discussed later in this thesis (→ Chapter 3). Additional tests (see Section A.2 of the appendix) show that without carefully chosen reweighting parameters, a pure (categorical) cross entropy loss could lead to shapes that almost only involve two bins in total, one close to zero, another close to one, with very limited statistics in between. This would make the derivation of scale factors over several bins per discriminator very difficult, if not impossible. When no reweighting is applied, the resulting model would do particularly well for the majority class(es), as those can contribute more to the loss during the training phase. Deploying the model for other samples that involve a different class distribution might then yield worse results if less abundant classes for the training are more abundant in the test set [97].

A second type of reweighting concerns the jet kinematics and follows the approach described in Ref. [8]. For demonstration, Fig. 2.6 shows the original distributions before the reweighting, split per flavour. 1D views of these histograms follow in Fig. 2.7, where the marginal distributions [2] for η or p_{T} alone are illustrated. For the different flavours considered, the p_{T} and η distributions differ. More specifically, jets originating from b quarks (especially those in subcategory bb) tend to have higher transverse momentum than non-b jets. Looking at the pseudorapidity distribution before any reweighting, udsg jets appear more frequent in higher η bins than those from the other categories. As with most checks as a function of fla-

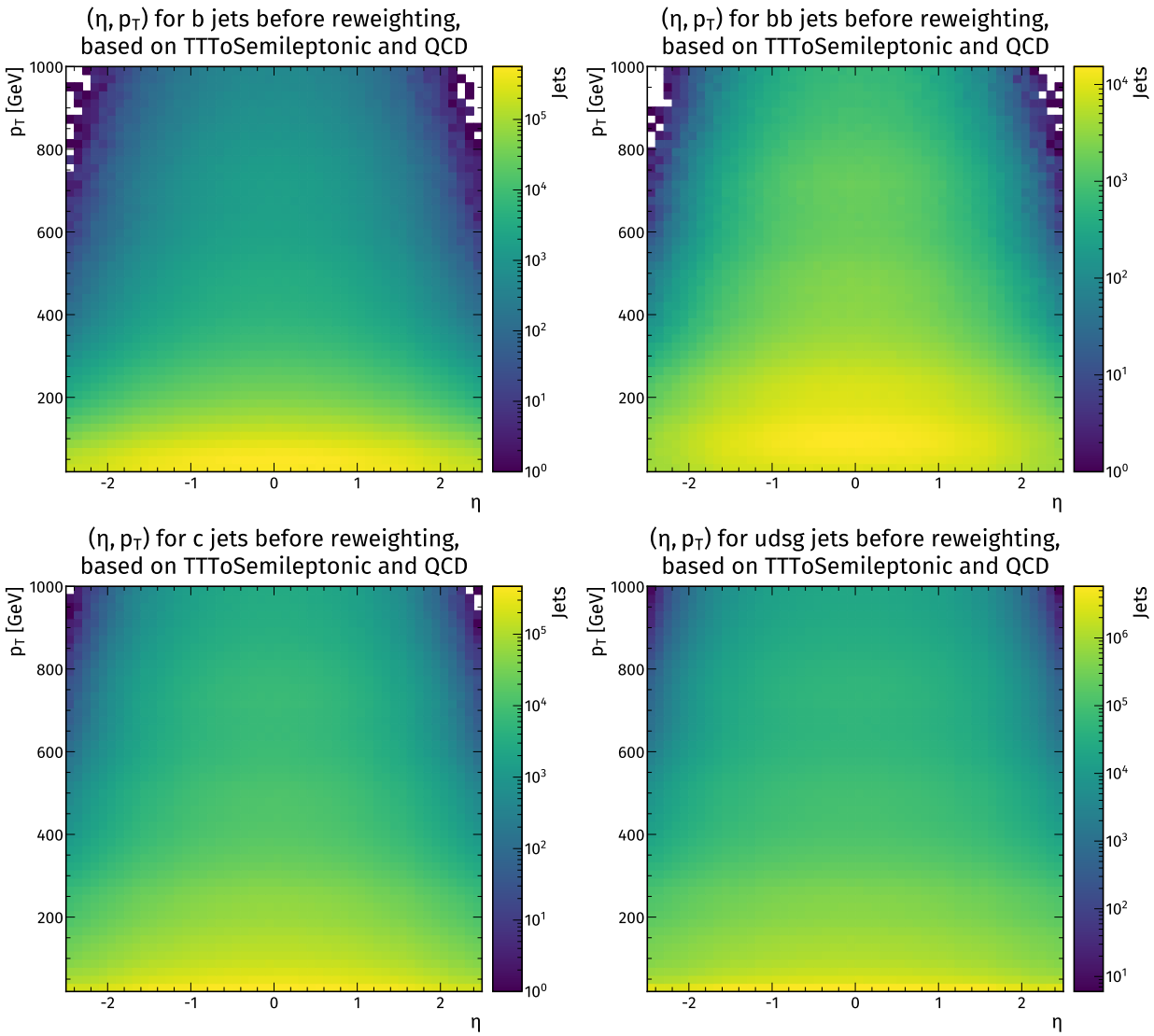

Figure 2.6: 2D (η, p_{T}) distributions for all jets before reweighting, split by flavour.

vour, those jets originating from c-quarks produce intermediate distributions between b and udsg jets. To avoid that the model learns unwanted features of the particular training sample regarding the imbalance in p_{T} and η bins due to the given distribution of flavours, a per-jet-reweighting is applied [8]. That way these two variables still contribute during the training, nonetheless, which is desired to incorporate possible correlations between the jet kinematics and other inputs [8, 94].

The reweighting should provide a good compromise between model performance and flavour independence of kinematic variables. For that reason, the sample weights have been generated using two slightly different strategies, allowing a comparison later. In both cases, all available jets are binned into a (50×50) histogram in (p_{T}, η) [94]. For the first attempt, the twodimensional target distribution is the average number of jets per individual bin, computed from four different histograms (one per flavour). The second variation aims for a flat target distribution over the full (p_{T}, η) phase space, minimizing the dependence on the jet kinematics even more. Additionally, both different target distributions ensure that a hypothetical resampling would produce equal numbers of jets in each of the four categories, i.e. the absolute numbers match, not only the relative fractions. For that purpose, another factor is multiplied that takes care of the flavour distribution. The weights are computed by taking the

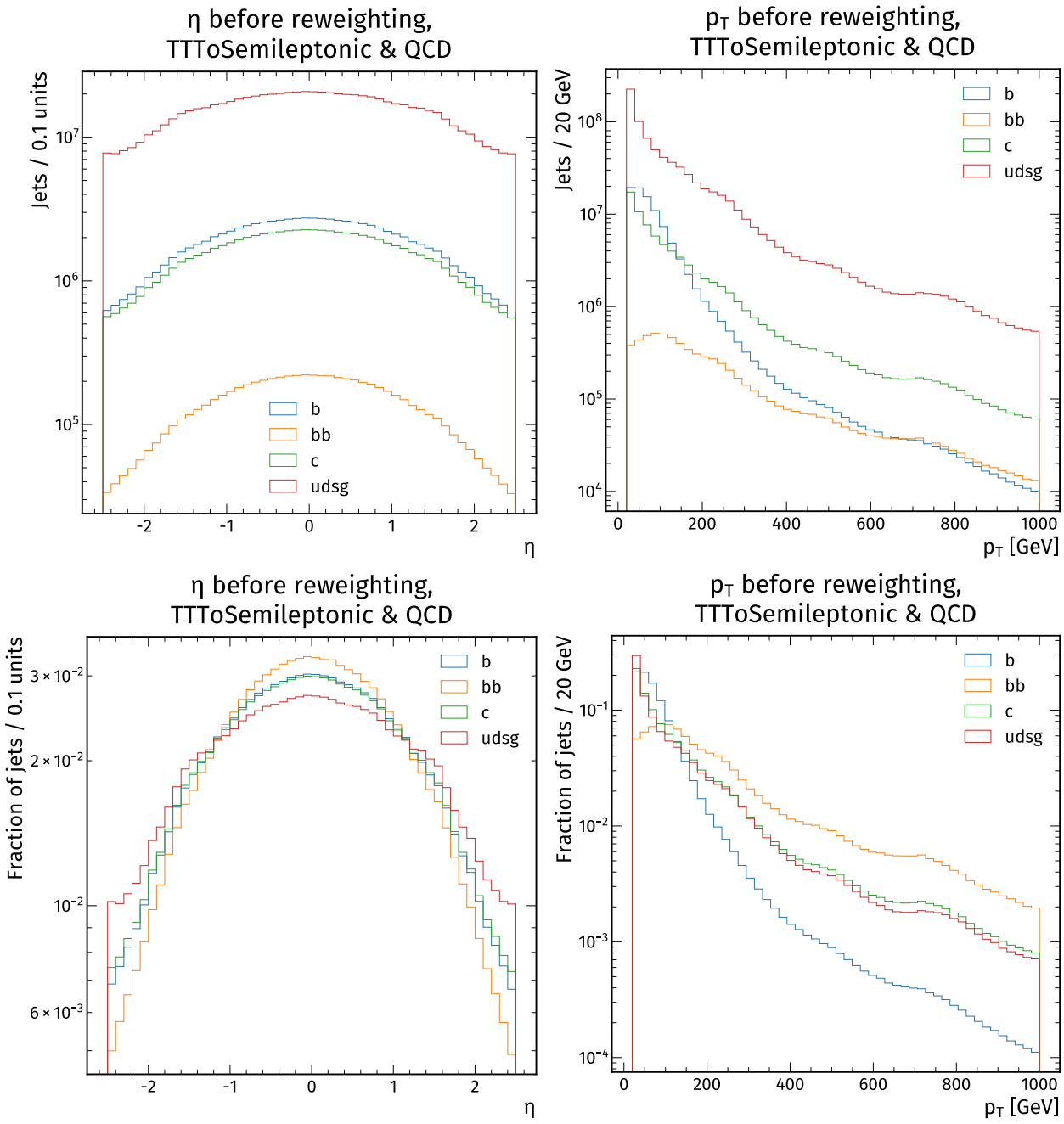

Figure 2.7: η distribution before reweighting (left) and p_{T} distribution before reweighting (right), split by flavour. Obtained by summing over p_{T} or η bins, resulting in marginal distributions in 1D for η and p_{T}. Absolute numbers are given in the first row, while the second row shows the relative fractions.

quotient of the target histogram and the original histogram for each flavour, respectively. Figure 2.8 contains the weights that correspond to the first reweighting strategy (per-bin-average over all flavours) in the left column, and the weights to obtain flat distributions in the right column. How these weights influence the pseudorapidity and transverse momentum can be seen in Fig. 2.9, again for both versions of the reweighting. A direct comparison of values is not possible, as averaging over four flavours produces much smaller initial weights, while the flattening can lead to large values, especially for high jet-p_{T}.

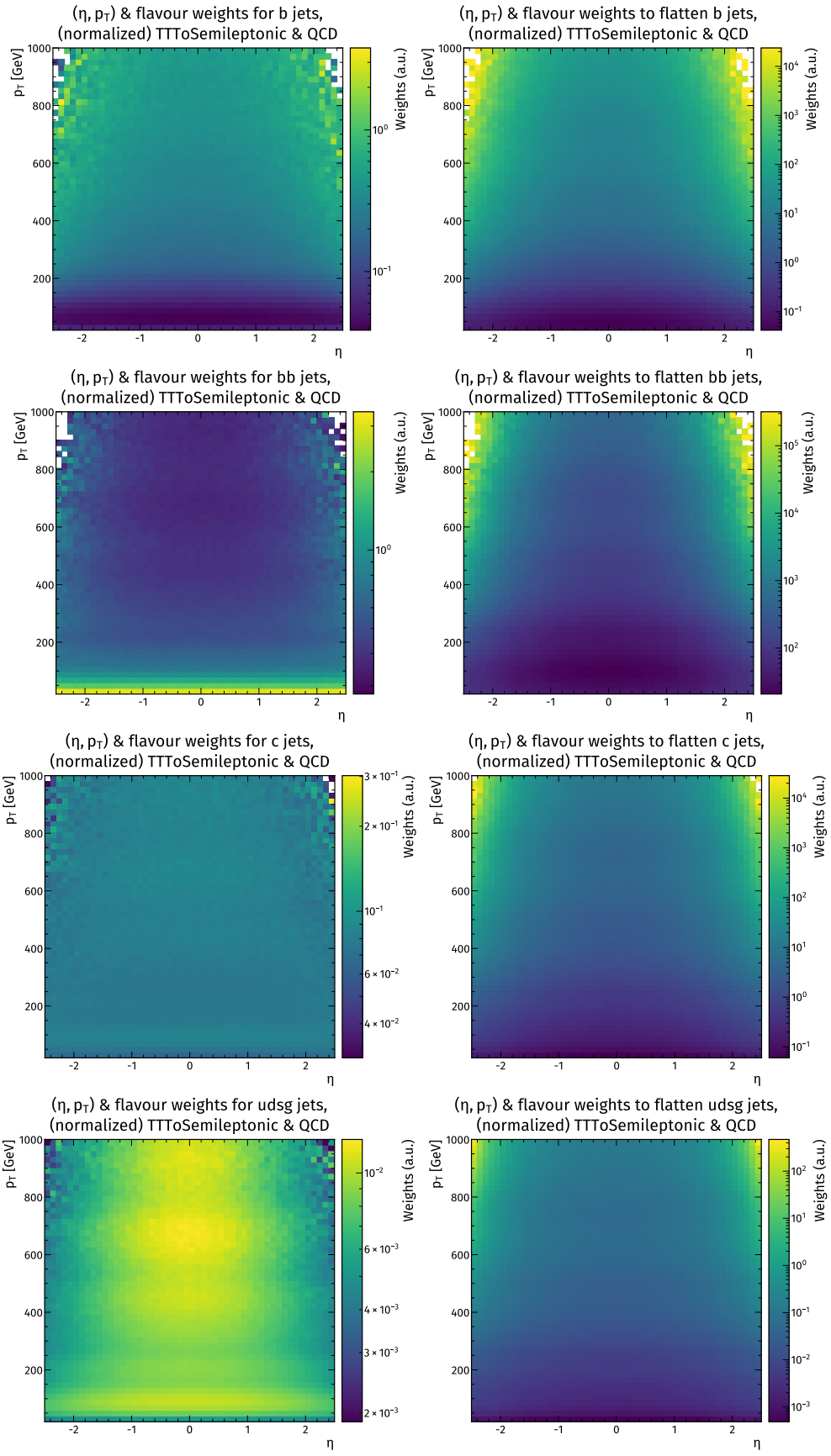

Figure 2.8: 2D (η, p_{T}) weights with two different reweighting approaches, split by flavour. The variant that takes averages per bin is displayed on the left, the one that produces flat distributions is shown on the right. Both columns reveal qualitative differences between the respective reweighting strategies. On the left side, the differences per flavour are clearly visible, on the right side, the differences are not as pronounced, because all flavours are "similarly" far away from a flat distribution. Therefore, absolute scales can not be compared directly.

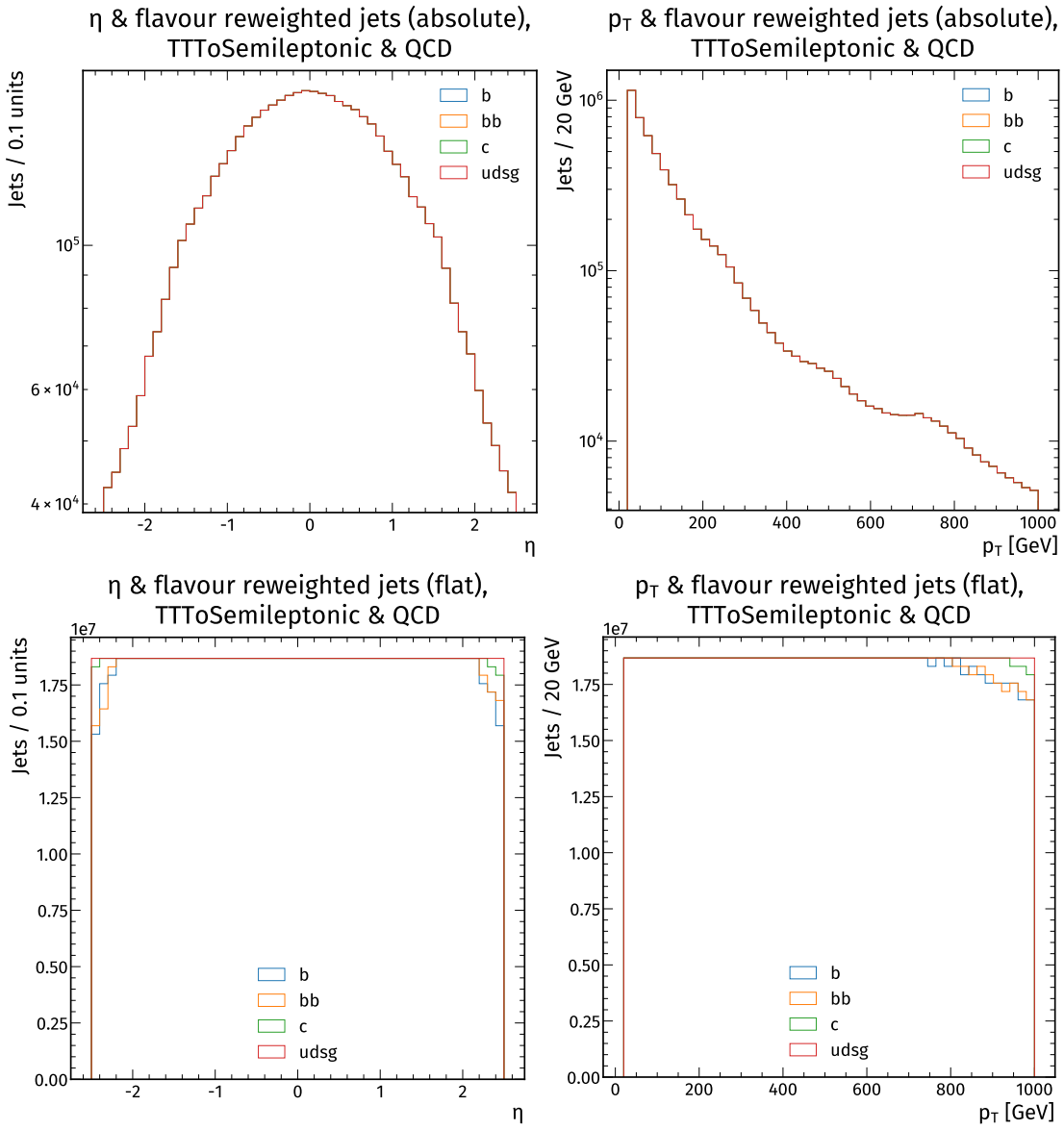

Figure 2.9: η distribution after reweighting and p_{T} distribution after reweighting, split by flavour, for two reweighting methods. The first row shows the results when per-bin-averaging is applied, the second row contains the resulting distributions that are flattened. The 1D views are obtained by summing the 2D histograms over p_{T} or η bins, resulting in marginal distributions for η and p_{T}. The scales can not be compared and shall only illustrate how the reweighting would influence the absolute numbers of jets in a hypothetical resampling that takes the distribution of different flavours into account. In the first row, the distributions for all flavours overlap (relatively and absolutely), which is why only one colour is visible. The flat distributions in the second row lack a small fraction of jets for the non-udsg classes, because some bins are empty in the original 2D histograms. This only happens for high jet-p_{T} and jet-η, though.

Empty bins in the 2D histograms are the reason for the tails at the edges of the individual 1D distributions in case of the flat reweighting.

The respective expected final 2D distributions (if resampling would be used during the training) are part of the appendix, where non-normalized absolute values per bin are given in Figs. A.1 and A.2. The absolute number of jets for the reweighted histograms is only to be understood as a theoretical value, the actual number of available training samples is not changed

in the current setup. The purpose of these plots is only to illustrate that the flavour or class imbalance has been corrected with this reweighting of the kinematic variables as well (effectively resulting in a identical distribution of flavours for the training). In the following section, the application of the weights during the training will be explained.

2.3.2 Using the weights during training

Loss weighting

In the setup considered for this thesis, the individual sample weights are used to modify the loss function for every batch. Depending on the weighting method to be applied, different histograms containing the weights are loaded. Every sample is checked for its flavour and the bin that corresponds to the jet-η and jet-p_{T} value. This "loss weighting" [52] is implemented by multiplying the loss with the sample weights [97]. To ensure that the reweighting does not alter the absolute scale of the loss, the element-wise products of `loss vector o sample weights` are first normalized per batch, and the resulting value for the batch loss is averaged to obtain a single scalar value. Then, the backpropagation runs with this weighted loss as usual. The behaviour of the loss, evaluated for different epochs during training, will be discussed in Section 2.4.3.

Resampling

An alternative way to incorporate the weighting during the training is to sample the jets based on the individual weights from a multinormal distribution. Getting the correct weight for each jet is done in the same way as described for the loss weighting, but then the drawn batches themselves incorporate the reweighting technique, which can be seen as an additional preprocessing step. This adds to the time (choosing the right indices) and space (duplication of samples) complexity, and might ultimately lead to overfitting for the highly imbalanced data set, as large numbers of jets would need to be drawn multiple times to match the correct total numbers [96]. This resampling is additionally implemented with an extension [98] to a generic PyTorch `Sampler`, but not considered further.

2.3.3 Focal loss

The previously described reweighting is a first step to correct the class imbalance, to reduce the dependence on the kinematic variables of the training sample and to spread out the tagger distributions between zero and one (see also Section A.2). However, large peaks at the edges might occur nevertheless. On the other hand, some jets might be intrinsically more difficult to classify than others, for example if variables are missing (see Section 2.2.4). To mitigate these problems, another type of reweighting is used. The idea of the application of the so called focal loss [99] is to give less weight to easily classified samples, while giving more weight to those examples that are difficult to classify. With this approach, it is possible to obtain rather spread out distributions and the training progress is more balanced instead of going too fast in one direction. The generic cross entropy loss function makes sure to do exceptionally well for the majority class already early during the training, yielding the sharp distributions, the other classes only follow later. Well-classified samples do have a decent contribution to the standard cross entropy loss, but with the focal loss, this behaviour is not expected [99]. In this case, the early drift into a minimum that optimizes for the majority class is slowed down by the higher impact of hard to classify samples. The formula behind this technique differs from the categorical cross entropy loss only by an additional factor $(1 - p_t)^\gamma$, where p_t is a placeholder for the *truth*-output probability p_i in case of a jet actually belonging to the considered category

i [99]. In the originally described case with only two categories (a binary classification prob-

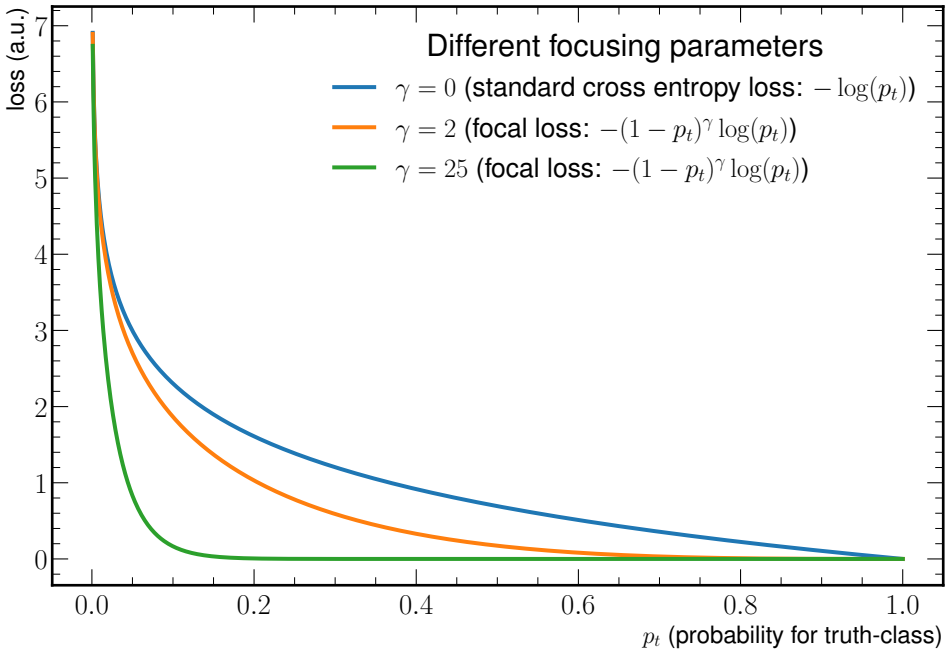

Figure 2.10: Dependence of the focal loss on the chosen focusing parameter. Using $\gamma = 0$ would restore the standard cross entropy loss function. Higher values like $\gamma = 2$ (see appendix) or $\gamma = 25$ (used for the investigations of robustness) suppress easy to classify samples. Adapted from Fig. 1 in Ref. [99], with slightly modified focusing parameters.

lem), p_t refers to the probability p of the output node if the sample is part of the ground-truth class (target $y = 1$) or to the probability of the complementary event $(1-p)$ when this is not the case [99]. Varying γ (a positive floating point number) controls the severity of this reweighting method and is called the *focusing* parameter [99]. From this definition it can be seen that the new loss is smaller than the original one obtained with the standard `CrossEntropyLoss` function [71], especially for samples whose predicted probability is high (see Fig. 2.10 and Ref. [99]). Now, p_t is bound between 0 and 1, large p_t values lead to $(1-p_t)$ values close to 0 (risen to a given power this becomes a tiny number), small p_t values result in $(1-p_t)$ values close to 1 (which stay close to 1 even with a specified exponent γ) [99]. It is possible to additionally multiply the loss with another factor based on the category. The multi-class implementation [100] of the focal loss offers four additional hyperparameters α_i, with $i \in \{0, 1, 2, 3\}$ that allow for manual class reweighting. In the present study, however, these numbers are not specified further and are kept at the default values (= `None`, i.e. manual weight $\alpha_i = 1$ for all i), as the reweighting by class, p_T and η is already handled separately.

2.4 Training

2.4.1 Storing the inputs and dataloading

As explained above, a large number of inputs is available for the training of the network. Also the storage of the model itself has to be taken into account. This, together with the targeted usage of gradients (with a computation graph) of input variables to create adversarial samples, demands a rather high memory consumption. A tradeoff has to be made that provides a high yield of training data but is feasible with the available hardware. As a single GPU is limited to 16 GB memory (the main CPU memory of one node being 192 GB [69]), it has been decided

to store the inputs with `Float16`-precision only. The efficient dataloading is carried out on a CPU, and only small batches are transferred to the GPU to perform the actual forward and backward passes.

Additional changes to a standard setup involve a custom `FastTensorDataLoader` [101, 102] that effectively slices arrays (or tensors) instead of looping over every element, which speeds up the training. Especially shuffling the samples is time consuming. The usage of randomly chosen indices (shuffling) for the minibatches provides a representative estimate of gradients if one would take the full data set into account [52]. In the current setup, one training epoch with the built-in `DataLoader` [103] with `shuffle` set to `True` would take about 48 hours, while the optimized version takes only 10 minutes. Also the batch size influences the time it takes for the training, apart from its impact on model convergence and the fluctuations of the loss over epoch. A rather high batch size of 1000000 jets is used, ensuring that the time saved with the GPU is not compensated by the time needed to switch the device for every small batch. The choice of the batch size is still much smaller than the total number of available training samples and thus ensures that the optimization does not directly run into local minima [52].

2.4.2 Choosing the training setup

Given the large number of hyperparameters and weighting methods available for the training, some decision for the training setup has to be made, with which the susceptibility to adversarial attacks will be investigated. For the majority of the checks that will follow, weights will be applied that target a balanced distribution of truth labels as well as a class-reweighting for the kinematic quantities p_{T} and η. That way the dependency on the flavour distribution is reduced during training, but the reweighting is not as severe as a flattening of the p_{T} and η distributions would be. For the focusing parameter, a value of $\gamma = 25$ has been chosen. As will be demonstrated in Chapter 3, this leads to smooth output shapes without excessively large bins at both ends of the distribution, but still results in a good performance. Several checks that show this behaviour for the picked parameters are provided in Chapter 3, where the nominal performance is explored. A fine-tuning of the hyperparameters still has to follow and some additional studies that tackle this problem are provided in the appendix for reference (see Section A.2).

For the parameters that are considered in this thesis, two separate trainings that incorporate either the basic approach with raw inputs only, or inject adversarial inputs already during the training phase, are performed. Besides this difference, the setup of both trainings is identical. For the adversarial training, the parameter $\epsilon = 0.01$ was found to be useful in terms of disturbing the baseline classifier, while the majority of the input variables are only modified slightly (with some unphysical exceptions, as will be discussed in Chapter 4). The abbreviations used in the subsequent figures (namely "p_{T}, η rew. (F.L. $\gamma = 25.0$)" and "p_{T}, η rew. (F.L. $\gamma = 25.0, \epsilon = 0.01$)") denote the reweighting technique (here: averaging the binned kinematic distributions over flavour) as well as the focusing parameter of the focal loss and indicate if the model has been trained with adversarial samples ($\epsilon = 0.01$) or not. Both models are trained for 200 epochs, for which the training history is discussed in the next section.

2.4.3 Convergence of loss over epoch

For imbalanced data sets, neither loss nor accuracy are representative metrics for the model performance, as their values are mainly driven by the majority class and do not offer information that considers the pairwise discrimination between different categories [52, 96, 97]. Therefore the training history as displayed in Fig. 2.11 only offers first hints on how well the

model has learned to distinguish the different flavours and will just serve as a check to exclude obvious under- or overfitting. More detailed evaluation techniques (of which many are independent of the flavour composition) follow in Chapter 3. For both trainings, the training as

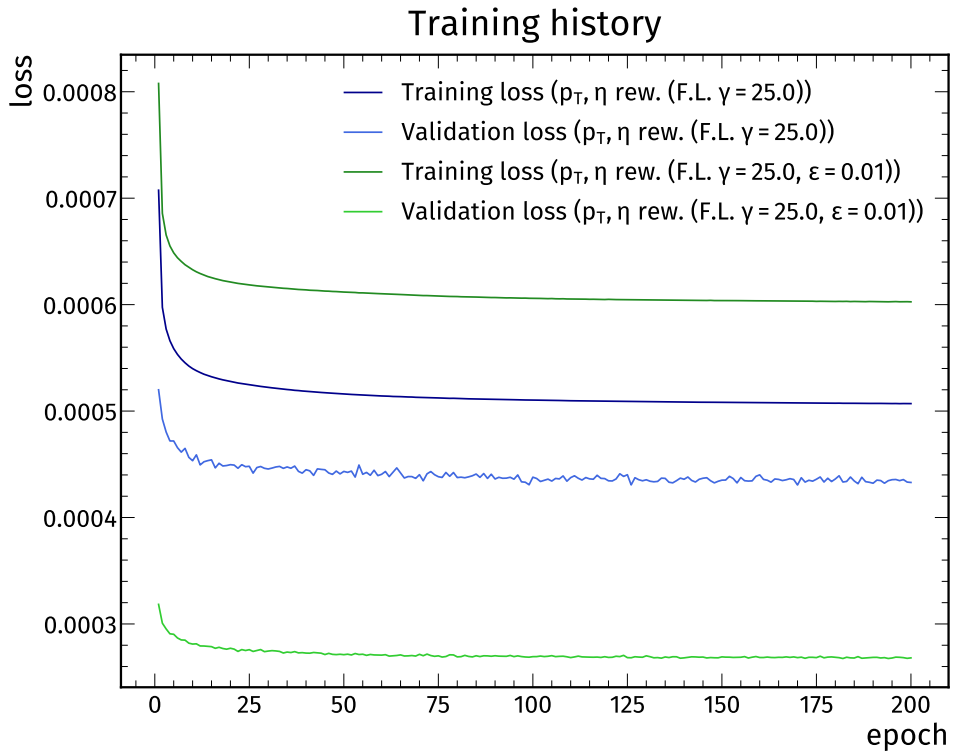

Figure 2.11: Training and validation loss over epoch. The blue curves represent the basic training and the green ones result from the training with adversarial samples. In both cases, the darker colour shows the training loss, while the lighter colour shows the validation loss.

well as the validation loss decrease very fast for the first approximately ten epochs. After this initial phase, the improvement with more epochs happens only slowly, and the curves eventually converge to their respective minima at about 200 epochs. From the behaviour of the objective function it is justified to omit early stopping as a regularization technique for now, however, with a dedicated search for hyperparameters it could become useful to determine a good point to stop the training process. In particular, no generalization gap is currently visible, which would be a sign of overfitting [2]. Only a very tiny residual slope for the training loss is noticeable for both models, while the validation losses basically only fluctuate and no large improvements are expected. All loss functions use an identical setup for the focusing parameter. The difference between the calculation for the training samples and validation set is that the training samples incorporate additional loss weighting (based on flavour, p_T and η, averaged per minibatch and later averaged per epoch), while the validation samples are inserted into the loss function "as is" without modifications of their individual weight, but using the model trained on loss-weighted training data. This can be seen as a method that allows conclusions on how well the network can generalize, as during its evaluation or real application, the jets would not be reweighted (there, the flavour is unknown). At the same time, this makes comparing the absolute values of the different losses (training versus validation) difficult. It is better to compare the training loss of the basic training with the training loss of the adversarial training (similar for the validation loss), because these two pairs use the same approach during their calculation and are quantitatively comparable. Then the conclusion would be that the training on adversarial samples overall seems to generalize better. For the basic training, it might be that some memorization effect of the training data sets in. Also

the fluctuations in the validation loss for the adversarial training are smaller than for the basic training. The stability or how flat the loss surface is will be discussed also when performing separate evaluations on raw and distorted test inputs and are part of the AI safety studies in Chapter 4. Before those investigations can be done, the following chapter shows the nominal performance, evaluated on undistorted samples.

3 Nominal performance

3.1 Tagger outputs and discriminators

The evaluation of the model on undistorted test inputs begins with the inspection of the tagger's output distributions, as shown in Fig.3.1. For all four output nodes that provide probability distributions P for a jet to belong to one of the categories b, bb, c or udsg [8], the figure allows a comparison between the DeepCSV outputs (summed over all flavours) and the custom classifier, split by flavour. The distributions obtained from the custom tagger at epoch 200 are less spread out than the ones provided by DeepCSV, an observation that holds true for all output nodes. However, the different reweighting techniques applied during the training definitely reduce the behaviour observed when applying no reweighting at all or only using small focusing parameters (see Fig. A.3, where the majority of the outputs are placed either at the 0 or 1 bin). Another difference between the newly trained model and DeepCSV is the definition of default values for the input variables (see Ref. [8] and compare with Section 2.2.4), with which a residual double peak structure can be explained. Looking at $P(\text{udsg})$ for the mainly investigated focusing parameter of $\gamma = 25$, this feature is present at around 0.35 (for $\gamma = 2$ this becomes visible at approximately 0.75, as seen in Fig. A.3). While for DeepCSV, the default values of the inputs are placed at zero (after scaling) [8], for the models that were trained for this thesis all default values are placed outside, but close to the main input distributions, as already discussed in Section 2.2.4. These can therefore contribute to sharp edges at both ends of the output distributions. Additionally, DeepCSV produces default values for the tagger outputs itself, if the selection criteria of at least one selected track and at least one secondary vertex are not met [8]. Originally, they are stored with -1 values, but to obtain reasonable ranges for the histograms, they are plotted at -0.04. For the custom tagger, such cases are not treated with default outputs, but the raw tagger scores are also calculated and shown for jets not passing the DeepCSV selection criteria. This accounts for approximately $2\,\%$ of all jets (see Figs. 2.2, 2.3 and 2.4) and will therefore only play a minor role in interpreting the tagger outputs. The non-defaulting of the corresponding outputs could be used in the future to test how well the reduction of the dependency on kinematic variables has worked, as these are practically the only valid quantities left for jets not passing the selection criteria and should not offer much discrimination power [8, 83, 94].

When using the tagger for an analysis, the distinction mostly happens between only two cases at a time instead of utilizing one-versus-all metrics like the raw tagger scores [8, 50]. The different tagger outputs are combined to form so called discriminators, where fractions of probabilities are constructed from the output nodes of the model [8, 48]. This reduces the multi-class classification to binary classification, where one flavour is compared against one other flavour. An additional merging that can be done is to put both the b and bb categories into one super-category that contains all jets that have originated from at least one b hadron, not splitting this up any further [8, 48]. To tag b jets in an analysis, it is possible to use the sum $P(\text{b}) + P(\text{bb})$ of both b jet and bb jet probabilities (defined as BvsC/L), but for c jets, using the probability $P(\text{c})$ alone is not a safe way to discriminate them from other flavours. This is the case because the properties of c jets naturally occur "between" those of b and light jets [8, 48]. The probabilities would only give a very limited view of the actual discrimination of c jets from other flavours, as this depends on the flavour distribution of the test sample. Frequently

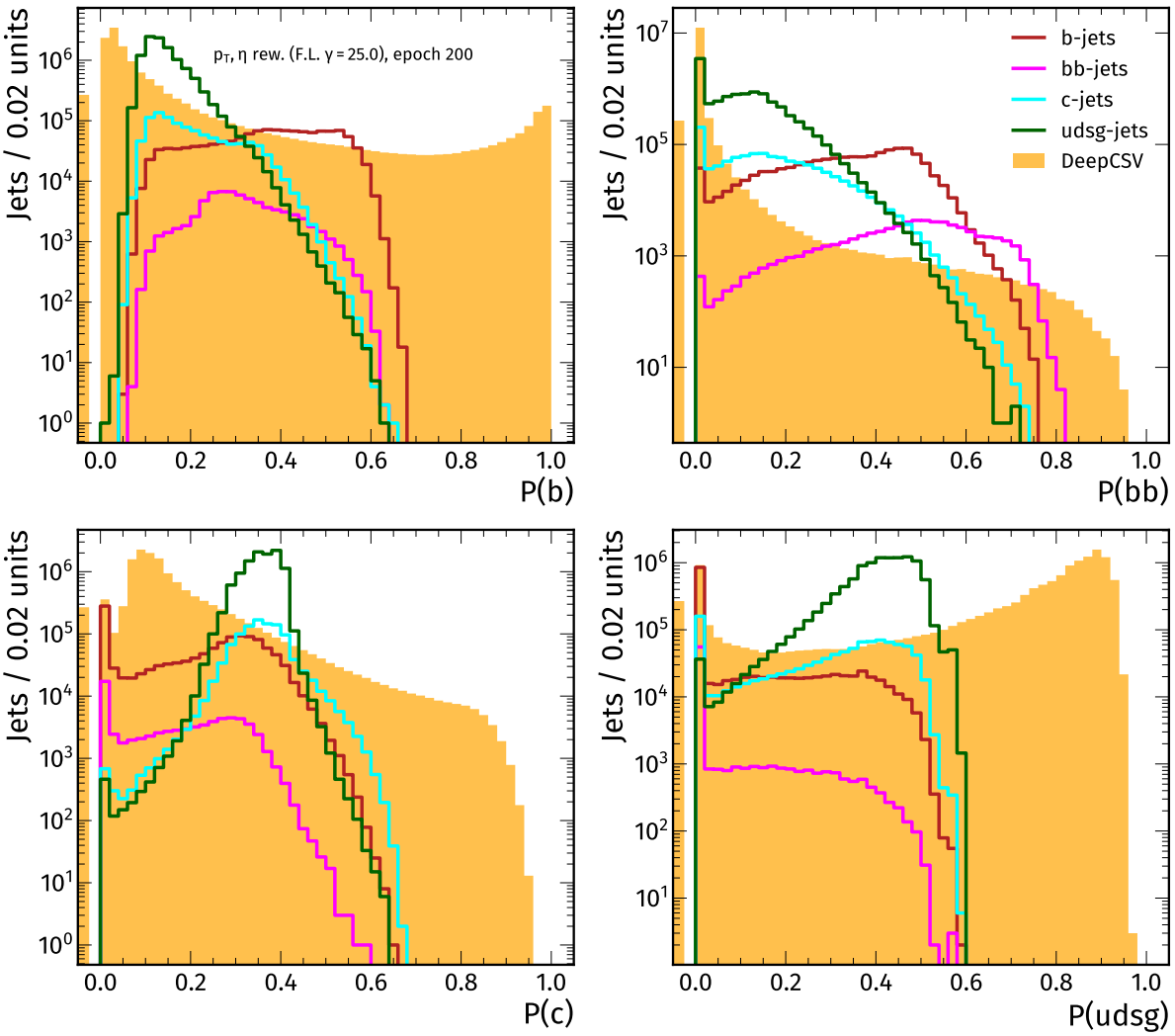

Figure 3.1: Tagger outputs, evaluating the basic training (split by flavour) and DeepCSV (summed) on test inputs. Each subfigure represents one output node of the model after 200 epochs.

used discriminators that evade this problem can be calculated as follows [8, 48]:

$$\text{BvsL} = \frac{P(\text{b}) + P(\text{bb})}{1 - P(\text{c})} = \frac{P(\text{b}) + P(\text{bb})}{P(\text{b}) + P(\text{bb}) + P(\text{udsg})}, \tag{3.1}$$

$$\text{BvsC} = \frac{P(\text{b}) + P(\text{bb})}{1 - P(\text{udsg})} = \frac{P(\text{b}) + P(\text{bb})}{P(\text{b}) + P(\text{bb}) + P(\text{c})}, \tag{3.2}$$

$$\text{CvsB} = \frac{P(\text{c})}{1 - P(\text{udsg})} = \frac{P(\text{c})}{P(\text{b}) + P(\text{bb}) + P(\text{c})}, \tag{3.3}$$

$$\text{CvsL} = \frac{P(\text{c})}{1 - (P(\text{b}) + P(\text{bb}))} = \frac{P(\text{c})}{P(\text{c}) + P(\text{udsg})}. \tag{3.4}$$

Calculating the discriminators shown in Fig. 3.2 introduces new difficulties that are related to default values as well as to floating point arithmetic. First, whenever one of the original output nodes is fixed to the default value of -1 (here, only for DeepCSV), also the discriminator should be defaulted to -1 [8, 48]. Second, decimal values are represented with binary numbers that are only approximate, with a limited precision. When (small) machine numbers are added or subtracted, "catastrophic cancellation" or "loss of significance" can occur, where the resulting value differs from the exact result more dramatically than what would have been

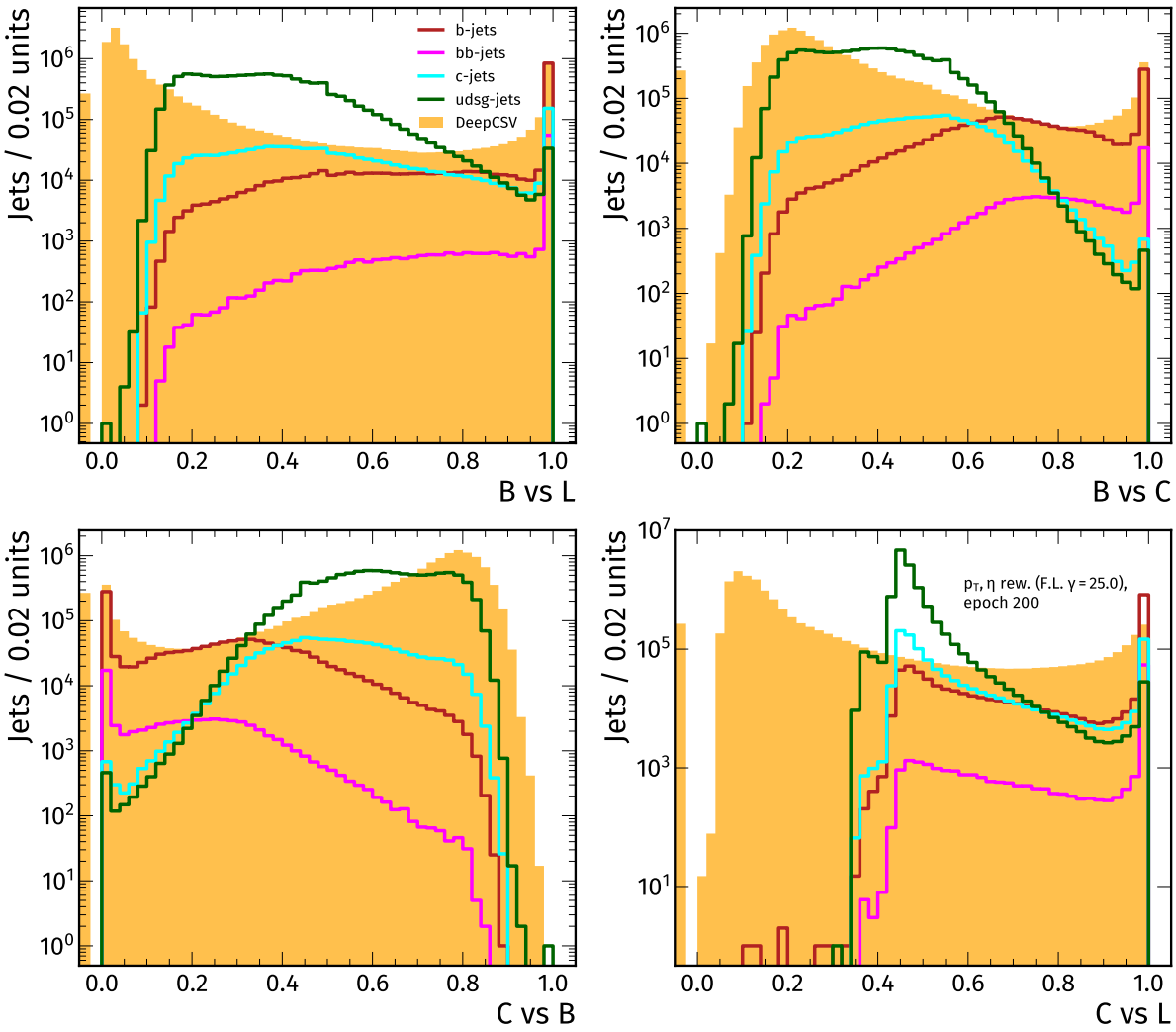

Figure 3.2: Discriminators, evaluating the basic training (split by flavour) and DeepCSV (summed) on test inputs. Each subfigure represents one discriminator between two flavours (with category b and bb merged into "B") after 200 epochs.

expected from error on the initial values [104, 105]. This feature is enhanced by the fact that due to storage constraints, the `Float16`-precision is used throughout all following investigations. To handle this effectively, out of the two mathematically equivalent expressions for each discriminator, the more numerically stable version is used. Experimentally, summing the available probabilities in the denominator has shown fewer occurrences of erroneous results than subtracting the probability of the counter-event from 1, therefore the expressions after the second equal sign in Eqs. (3.1) to (3.4) are preferred. In addition, all discriminators that would exceed values of exactly 1 due to rounding errors are truncated to stay within the interval $[0, 1)$. When interpreting the discriminator shapes in Fig. 3.2, it can be seen that the distributions for BvsC and CvsB are mirrored (excluding the defaults), this is due to the fact that they are not independent of each other [48]. With the definitions in Eqs. (3.2) and (3.3), this feature can be explained by noting that BvsC + CvsB = 1. Although the tagger distributions in Fig. 3.1 differ noticeably in their shapes between the custom tagger and DeepCSV, the discriminators are much more similar. A remaining difference is the cutoff of the distribution for both discriminators that involve the udsg class, which for example occurs for CvsL at about 0.35 for the custom tagger, while the distribution obtained from DeepCSV just reaches 0. In that sense the reweighting established in Section 2.3 with the usage of a large focusing parameter

$\gamma = 25$ is able to compensate for the distributions being pushed to one of the two extremes at 0 or 1 (see Fig. A.4 in the appendix), but a new issue, the cutoff, appears. The aim for further investigations is to reduce the residual sharp cutoffs with the help of hyperparameter-tuning, more balanced training data for all classes or the choice of different default values for the inputs, such that the distributions resemble those of DeepCSV.

3.2 ROC curves and AUC

With the help of Receiver Operating Characteristic (ROC) curves it is possible to visualize the performance of (binary) classifiers while adjusting the discrimination thresholds [106, 107, 96]. The classifier response, as displayed in the previous section in Fig. 3.1 (for tagger outputs) or Fig. 3.2 (for the actual discriminators), is translated into a twodimensional representation by varying the decision cutoff to discriminate between two categories. The corresponding true positive rates (TPR) as well as the false positive rates (FPR) are recorded, which are then taken as the x and y axis, respectively [96]. Their formal introduction requires the explanation of related quantities as an intermediate step. A definition of the different cases or outcomes for binary classification, arranged in a so called confusion matrix [96], is given in Table 3.1.

Table 3.1: Confusion matrix of a binary classification problem, where predictions are compared against the actual classes [96].

	Positive prediction	Negative prediction
Positive class	**True Positive** (TP): correctly identified examples of the positive class as part of the positive class	**False Negative** (FN): wrongly identified examples of the positive class as part of the negative class
Negative class	**False Positive** (FP): wrongly identified examples of the negative class as part of the positive class	**True Negative** (TN): correctly identified examples of the negative class as part of the negative class

For all four cases, the corresponding quantities TP, FN, FP and TN are either considered as absolute numbers of instances or fractions of the whole data set. Fractions are not necessary when these values are again used to calculate derived rates, as the total number of samples cancels out. From these definitions, various performance metrics can be constructed, which will be introduced in the following. It should be noted that due to a large class imbalance in the available data set, generic measures like accuracy should not be used to determine the performance [97]. The definition

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \tag{3.5}$$

might overconfidentially report high values for a classifier's performance although the classifier is not able to correctly identify a single element of the minority class, while assigning the majority class to all examples (including both the actual positive and actual negative classes) [96]. Instead, the investigation can be limited to one actual class at a time to quote the percentage of those samples being correcly classified or misclassified, individually per truth class. This differs from the approach above, where the denominator to which the instances are compared to is equal to the total population [96].

These quantities alone would again not be appropriate to indicate the performance of the classifier with a single metric, as it should do well on all classes. Therefore the combination of two of those measures (namely TPR and FPR) will be used as the two axes for the ROC

Table 3.2: Derived performance metrics for two-class problems, measuring rates of correctly classified or misclassified samples [96].

True Positive Rate (TPR): fraction of positive instances correctly classified as positive $$TPR = \frac{TP}{TP + FN}$$	**False Negative Rate** (FNR): fraction of positive instances misclassified as negative $$FNR = \frac{FN}{TP + FN}$$
False Positive Rate (FPR): fraction of negative instances misclassified as positive $$FPR = \frac{FP}{FP + TN}$$	**True Negative Rate** (TNR): fraction of negative instances correctly classified as negative $$TNR = \frac{TN}{FP + TN}$$

curves introduced above [97]. Instead of TPR and FPR, the terms tagging efficiency and mistagging rate are used to match the naming scheme of e.g. Refs. [8, 48]. To again reduce this to a single scalar quantity, the area under the ROC curve (AUC) is calculated subsequently and will be a key instrument to measure the performance of the classifier [97].

Alternative ways to report the performance make use of precision-recall (PR) curves [106], which utilize the recall or sensitity (equal to the TPR) as well as the precision (equal to the fraction of true positives compared to all positive predicted samples) and provide a less optimistic picture when there is a large class imbalance [97]. The fact that all models, setups and later also distortions are only compared against each other using identical evaluation metrics allows conducting all investigations solely with the ROC curves.

A comparison of different hypothetical classifiers is given in Fig. 3.3, where discriminators and ROC curves are visualized. Additionally, the corresponding curve for a random classifier is added as well. For heavy-flavour tagging, the convention is to display the efficiency on the x axis (with a logarithmically scaled mistagging rate on the y axis) [48, 8], while classically, the axes are flipped and both use linear scales [106, 107]. With the logarithmical axis for the mistagging rate, it is easy to identify the different working points "loose", "medium" and "tight" ($10\,\%$, $1\,\%$ and $0.1\,\%$ mistagging rate [8]) in the figures. Only for the illustrative figure, both styles are placed next to each other, for reference, but all succeeding ROC curves use the b-tagging convention. For the classical visualization (TPR on the y axis), ROC curves that extend further to the upper left correspond to better performing models, while for the visualization with flipped axes, lower right means better performance. A hypothetical perfect performance would correspond to an AUC of 1 [106, 107]. If the originally calculated AUC is smaller than 0.5, a flip of the axis for the discriminator also mirrors the ROC curves at the bisector, such that the resulting AUC would be larger than 0.5 again. This is plausible, because the discriminating power of the classifier is independent of the direction into which the two classes are separated. Therefore the worst case actually corresponds to an AUC of 0.5 (and not to 0), where there is no such separation between the classes [106, 107].

3.2.1 Performance after 200 epochs

The output nodes of the model correspond to probabilities for a jet to belong to one of the four categories b, bb, c or udsg. These probabilities can in principle be used to construct ROC curves that compare one flavour to all flavours. The limitation of the inclusion of all flavours is that the denoted performance metrics depend on the flavour composition of the test sample. The multi-class problem needs to be reduced to the binary case in order to quote

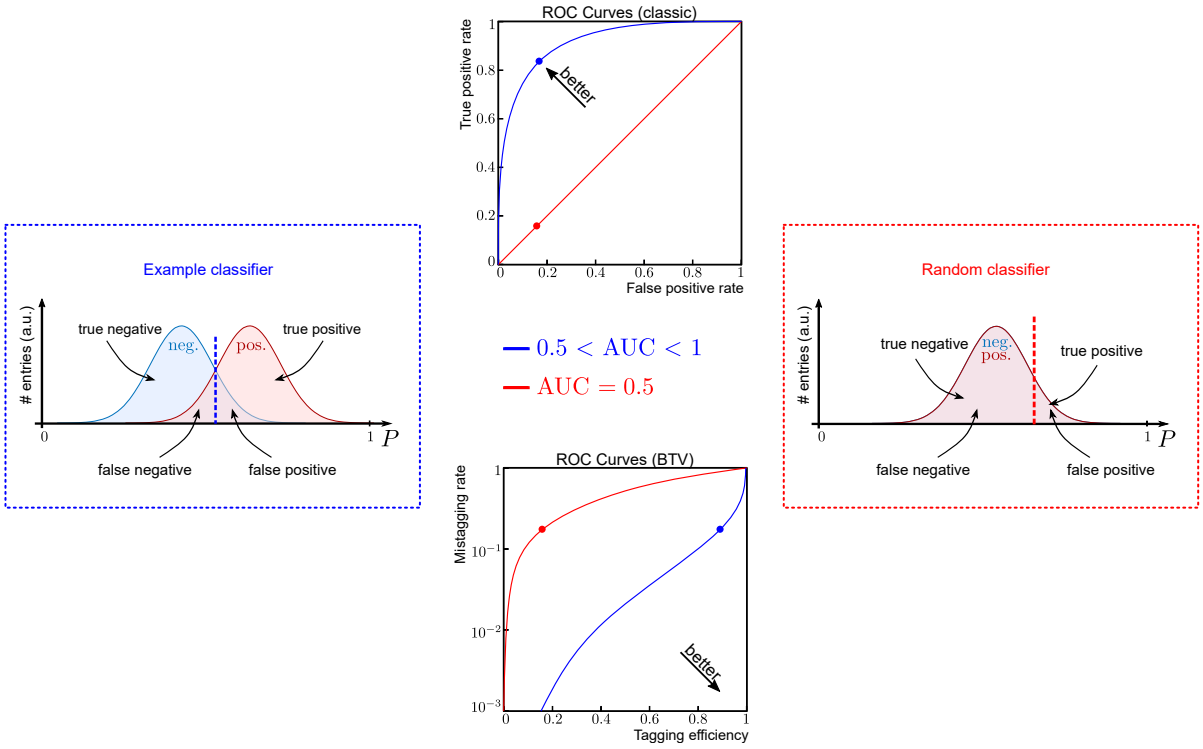

Figure 3.3: Comparison of two hypothetical classifiers, showing the discriminators as well as the corresponding ROC curves. The top ROC curves use the classical axes style and labelling, while the bottom ROC curves adapt the BTV-style. Compared to the example on the left with reasonable tagger performance, the right illustration shows a random classifier. Some possible cutoffs (or thresholds / working points) are given by dashed lines (discriminator shapes) and points (ROC curves). Gaussian distribution adapted from Ref. [61].

how well the classifier is able to separate b from light jets, for example (see also Section 3.1). Otherwise the performance measure can not report the individual separation powers and will change when the fractions of flavours are modified. Figure 3.4 shows ROC curves of individual output nodes for the model that was trained on undistorted inputs (basic training), evaluated after 200 epochs on unseen test data. For the reasons discussed above, the more important ROC curves associated with the four discriminators are displayed in Fig. 3.5, as universal performance metrics that do not depend on the flavour composition of the data set are preferred [8, 48]. All subsequent comparisons for different checkpoints of the training or for different adversarial attacks are done with the BvsL discriminator, giving rise to the title of this thesis, which focuses on b-tagging. Attempting to interpret the ROC curves of the type "X versus All" results in the conclusion that the performance of the custom tagger for b, bb and udsg jets is comparable to that of DeepCSV. This includes the performance on the class with the lowest abundance in the data set, namely the bb category. Only for c jets (c versus all jets), DeepCSV shows a significantly higher performance, where the AUC of 0.5359 reveals that the custom tagger is only slightly better than a random classifier. However, the identification of c jets is more difficult than identifying b jets, because the properties of c jets are situated between those of b and light jets [8, 48]. Due to this and the fact that c jets only constitute 8.1 % of the whole training sample (see Table 2.5), a rather low performance is expected. Another trend for the different categories is that at low mistagging rates, the custom tagger performs slightly worse than DeepCSV, while for mistagging rates at or above the loose working point, the custom tagger improves such that both curves start to overlap or to intersect. An example that shows how different hyperparameters can influence the performance is given in the appendix, where the focusing parameter is reduced (see Fig. A.5). This is another point that supports

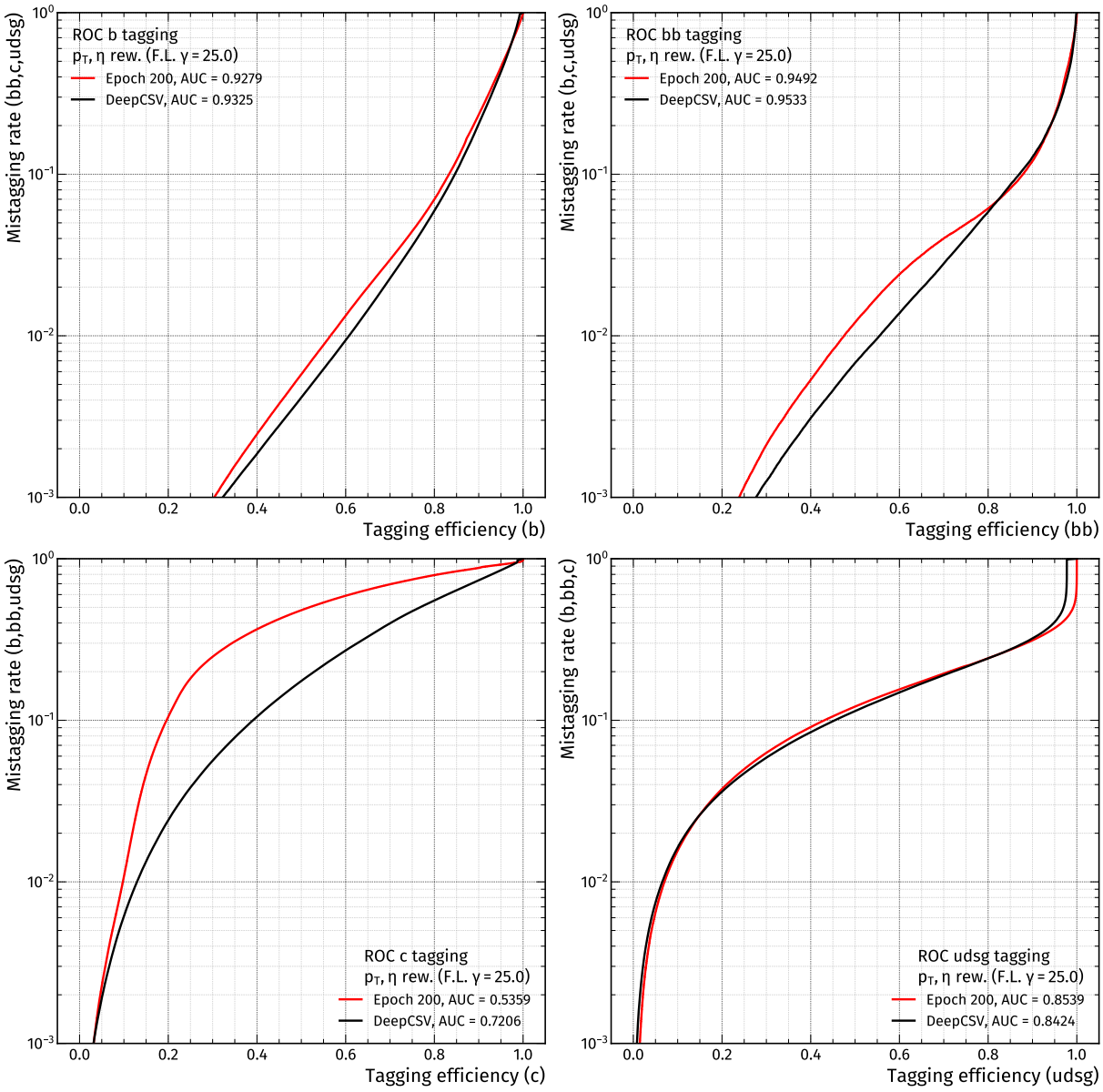

Figure 3.4: ROC curves when discriminating each flavour (b, bb, c, udsg) against all other flavours. This metric depends on the flavour distribution present in the test sample. Therefore, it is not safe to measure the performance with this method. The custom tagger is shown in red, the DeepCSV classifier is visualized with black lines.

the idea of systematic hyperparameter-tuning for future investigations.

For the discriminators introduced in Eqs. (3.1) to (3.4), the ROC curves are only generated from the classes under investigation, e.g. the BvsL–ROC curve is only computed from true b, bb and light jets, excluding c jets, the other cases are treated similarly. Also here, default values and numerical precision need to be handled with special care and jets that fall in either of the special cases mentioned earlier are excluded for this evaluation step. The ROC curves related to the four discriminators discussed above show that the newly trained model and DeepCSV offer a comparable performance for the separation of two chosen flavour categories each. For the BvsC and CvsB ROC curves, at all displayed mistagging rates, DeepCSV is slightly better than the custom tagger. Although the shapes of the ROC curves for BvsC and CvsB are different (BvsC does not show the turning point that is present for CvsB), the AUC is equal. This matches the definitions in Eqs. (3.2) and (3.3), where CvsB $= 1 -$ BvsC, which results in ex-

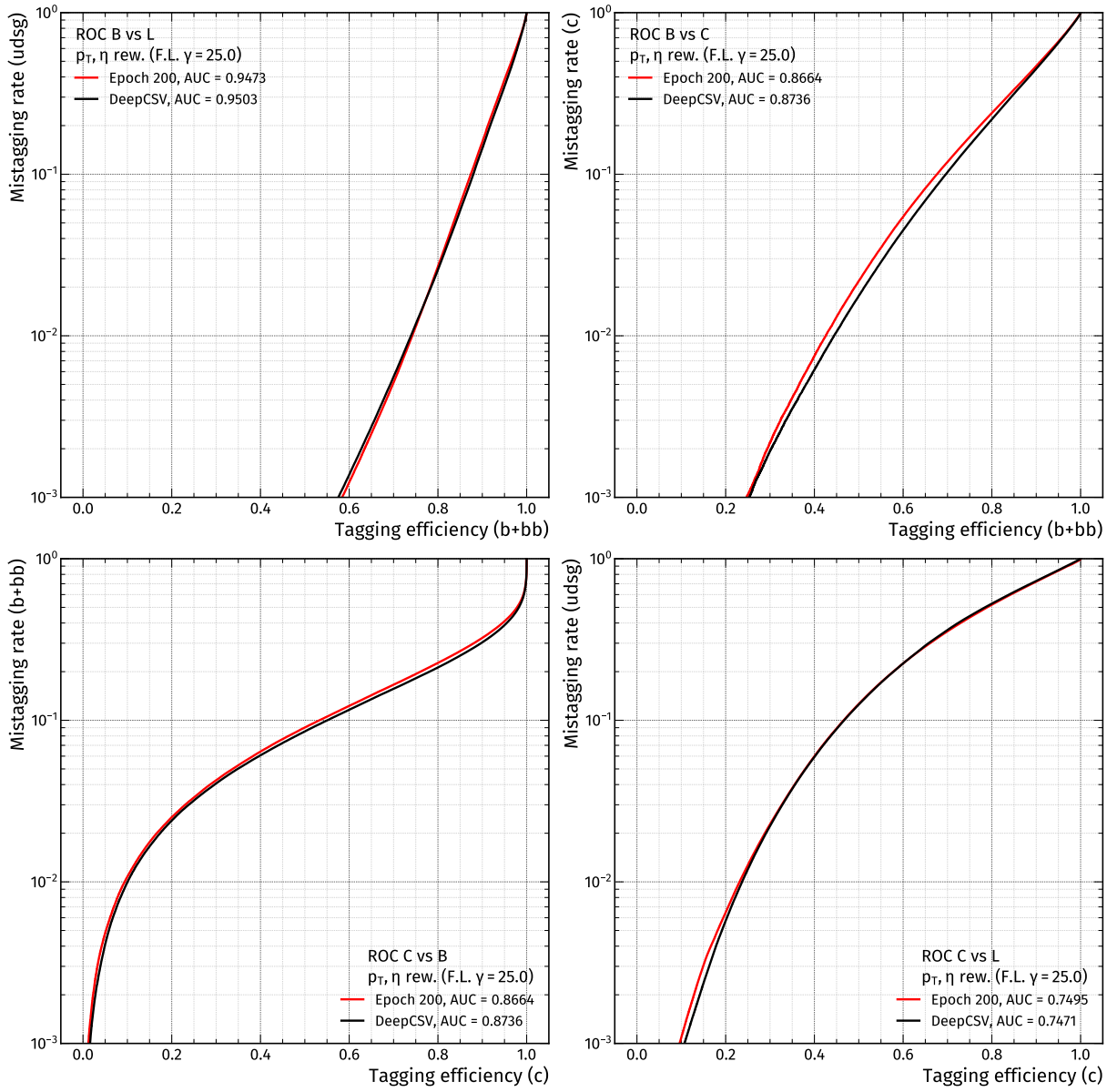

Figure 3.5: ROC curves for four different discriminators: BvsL, BvsC, CvsB and CvsL. The custom tagger is shown in red, the DeepCSV classifier is visualized with black lines.

actly inverted discriminator shapes and flipped thresholds. The AUC-ROC must be equal in that case, by construction. For the discrimination between b and light jets, the custom tagger and DeepCSV offer a similar performance, their ROC curves intersect at about $75\,\%$ tagging efficiency for b jets. Above approximately $1\,\%$ mistagging rate (udsg jets misclassified as b jets), DeepCSV is slightly better, below this medium working point the custom tagger has a marginally better performance. For CvsL, the intersection occurs approximately around the loose working point ($1\,\%$ mistagging rate for udsg jets) at $\approx 50\,\%$ c-tagging efficiency. Here, DeepCSV performs better in the low mistagging region and slightly worse for high mistagging rates. However, especially for both discriminators that involve light jets, the differences between the custom tagger and DeepCSV are so small that this could also be summarized as an equal performance, as such differences would be absorbed by systematical uncertainties [8, 48].

3.2.2 Dependence of performance on the number of epochs

In order to investigate how the performance of the classifier changes during the training phase, a checkpoint of the model is stored after every completion of an epoch. These checkpoints are then used to run the inference with unseen test samples, and the resulting ROC curves can be compared against each other. In Fig. 3.6, the BvsL–ROC curves are presented for epochs 1, 5, 10, 50, 100, 150 and 200 that span the whole training phase with small intervals at the beginning and larger intervals when the model shows convergence. The performance gain is largest at the beginning, for example the difference in AUC between epoch 1 and 5 is at the percent level. After the model has seen all training samples a couple of times, the performance increases only to a lesser extent, until the changes become practically invisible between epochs 150 and 200. This suggests that more iterations over the full data set with the current setup of hyperparameters are likely not going to improve the performance further. The checkpoints as well as the recorded AUC values will be used again when investigating the performance on distorted inputs, probing the robustness to mismodelling in the next chapter.

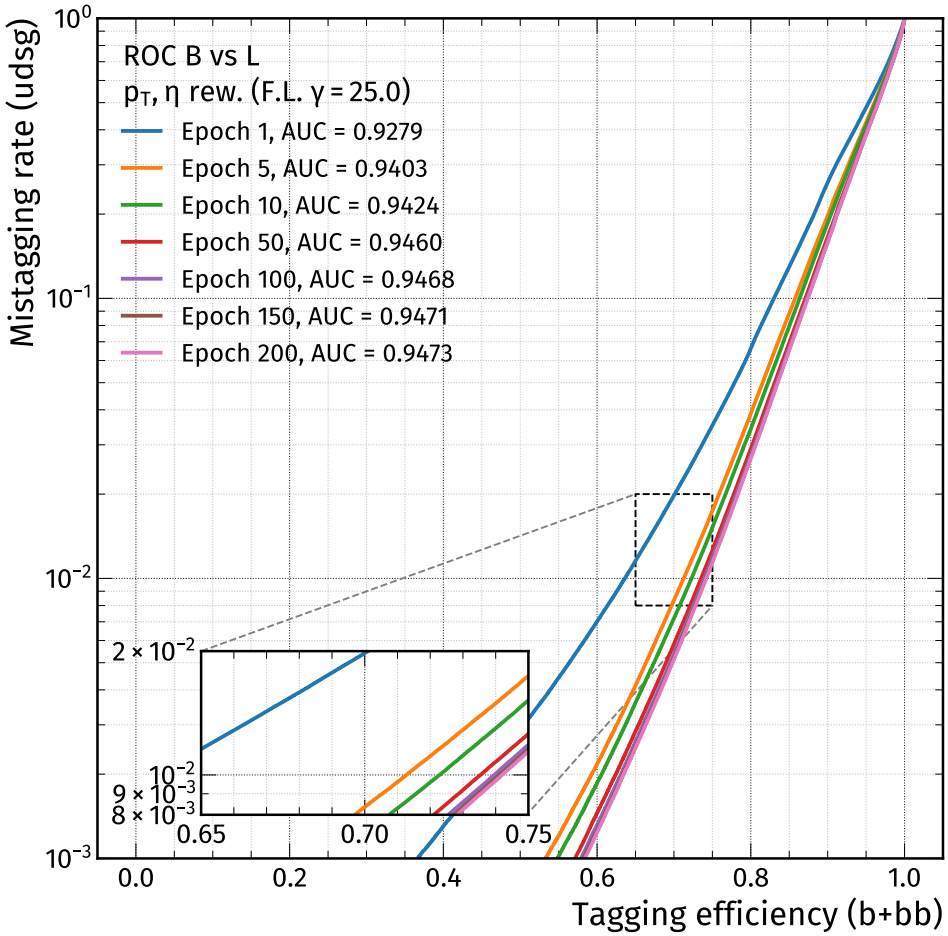

Figure 3.6: ROC curves for the BvsL discriminator, evaluated at different checkpoints of the training. The area under the ROC curve (AUC) increases with more epochs, but the gain in performance becomes smaller as the model converges.

4 Robustness to mismodelling

This chapter studies the impact of adversarial attacks as well as random noise on the performance of the custom jet flavour tagger. Apart from the evaluation metrics introduced in Chapter 3, also the input features themselves are investigated in detail, to reveal the effects on physical observables and to showcase possible limitations of the current approach. Where applicable, comparisons between the basic and adversarial training highlight the different response of a highly susceptible, versus a more robust neural network.

4.1 Distorted input shapes

As a first step to understand the effect of the distortions applied to the original inputs, a selected set of variables is chosen to show the raw as well as the modified distributions. For various features, the number of entries per bin span several orders of magnitude, therefore a logarithmic y axis is utilized for the histograms. For non-integer variables, 100 bins offer a granularity that allows detailed comparisons between raw and slightly distorted inputs. Default values are not modified and are placed outside the main part of the distributions. In certain cases, default bins are not visible at all due to the chosen range or scale for the x axis, if this improves the visibility of the interesting effects induced by the manipulations in the central part of the distribution. As will be discussed later, the integer variables are not distorted by the attack or the noise term, because the small distortions and necessary discrete steps of integer size contradict each other. For several variables, small differences in the occupancy of bins between raw and distorted inputs are hard to detect, therefore the ratio plot attached below the main histogram offers additional information. For every bin, the number of jets after the distortion $n_{\text{distorted}}$ is divided by the number of jets prior to the distortion n_{raw}. Assuming a Poisson error for the number of entries in the histogram [15] ($\sigma_{\text{raw}} = \sqrt{n_{\text{raw}}}$ and $\sigma_{\text{distorted}} = \sqrt{n_{\text{distorted}}}$) and using the Gaussian error propagation, the error on the ratio can be obtained as follows:

$$\sigma_{\text{ratio}} = \sqrt{\left(\frac{1}{n_{\text{raw}}}\sigma_{\text{distorted}}\right)^2 + \left(\frac{n_{\text{distorted}}}{n_{\text{raw}}^2}\sigma_{\text{raw}}\right)^2}. \tag{4.1}$$

Due to the large number of selected jets in the investigated test data set, the statistical uncertainties on the ratios will be invisible for most input variables. The control plots mainly show the impact of the FGSM attack (see Section 4.1.1), only few examples are included for the distortion with Gaussian noise to demonstrate the differences when splitting the distributions by flavour (see Section 4.1.2, where the non-systematic, random noise is visualized as well). A selection of additional features is presented in the appendix. The bin size as indicated on the y axis label has to be rounded for all cases where the selected x axis range has no finite decimal representation.

4.1.1 Inputs, summed over all flavours

Not all features can be shown here, therefore a selection has to be made that demonstrates examples where the attack stays undetectable and others where the impact would be unphys-

ical. The implications of the different visibility of the adversarial attacks for different features will be discussed in Section 4.1.3, this section mainly focuses on describing the behaviour.

As a starting point, the first features shown here in Fig. 4.1 are not affected too much by the FGSM attack, either because the distribution is originally rather flat and the feature has negligible discriminating power or because it is an integer variable (no changes at all, by construction). For the pseudorapidity, the agreement between raw and systematically distorted

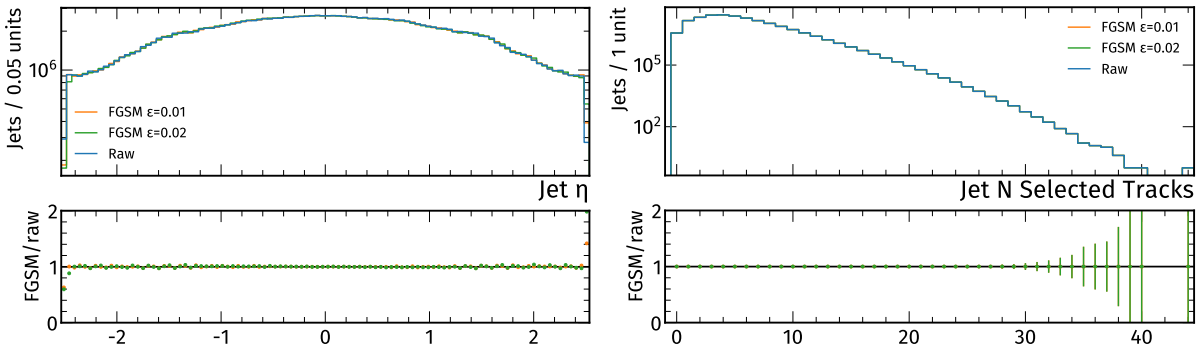

Figure 4.1: Distribution of raw and distorted inputs, for features where the impact of the adversarial attack is marginal or where the attack does not modify the distribution at all, by construction. The left figure shows the jet pseudorapidity with slight distortions, while the right figure demonstrates an example of an integer variable that is left unchanged. For the adversarial attacks, the parameters are either moderate ($\epsilon = 0.01$) or rather large ($\epsilon = 0.02$).

inputs is high, even for the large parameter ($\epsilon = 0.02$) for the FGSM attack, only at the edge of the distribution some jets get placed outside the originally chosen tracker acceptance. In the central part, entries in the bins might get translated in such a way that they switch between neighbouring bins of approximately equal height, which can not be visualized with the current method and granularity. As mentioned before, for an integer variable like the number of selected tracks in the jet, all distributions are exactly the same, for raw and distorted inputs.

More features are displayed in Fig. 4.2, for example the first row shows the same type of track variable for the first and sixth track (ordered by 2D signed impact parameter significance), namely the distance between the track and the jet at their point of closest approach. Due to the selection criteria introduced earlier, the fraction of defaults is much higher for the sixth track than for the first track. In both cases, the default bin is located close to the main part of the distribution, next to the minimum of each distribution. For the sixth track, there is a clear slope for the ratios, most tracks that originally show values close to zero are shifted towards the negative region, which seems to average out when the raw distribution flattens. This behaviour is not as pronounced for the first track, where the overall number of non-defaulted values is higher and the distribution appears to be more smooth (flat). Common to all tests is that larger values of ϵ increase the visibility of the attack, both in the input shapes, as well as in the ratio plots, as expected. With the flight distance significance (3D) value, there is a feature that should originally offer some discriminating power between the different flavours, because this quantity is related to the lifetime of the decaying hadron (see Section 1.4 and Ref. [8]). Indeed this feature is affected systematically when the FGSM attack is applied, significances starting from approximately 5 are shifted mostly to higher values. This could be related to the fact that the FGSM attack shifts the raw inputs into regions that are atypical for a given flavour, because this would confuse the network. Here, the shift is mostly due to light jets placed at larger flight distance (sig.) values, a feature that would normally correspond to heavy-flavour jets due to the longer lifetime of heavier hadrons. The fourth quantity (signed impact parameter significance of the first track in a transverse projection) shows a kind of wavelike pattern at significances close to zero. Some of the jets with original significances of

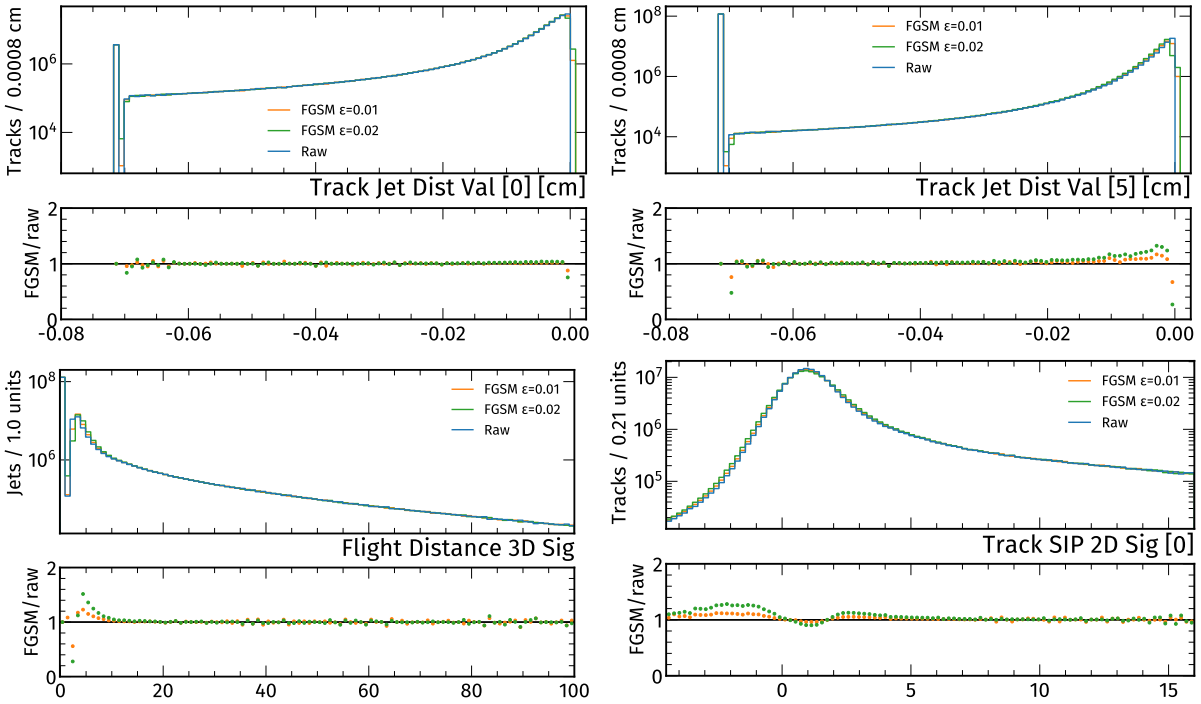

Figure 4.2: Distribution of raw and distorted inputs, for features where the impact of the adversarial attack is visible and results in specific patterns for the ratios. The top row shows the distance between the first (on the left) and sixth track (on the right) and the jet axis at their point of closest approach, while the second row depicts the SV 3D flight distance significance on the left and the the 2D signed impact parameter significance of the first track on the right. For the adversarial attacks, the parameters are either moderate ($\epsilon = 0.01$) or rather large ($\epsilon = 0.02$).

approximately 1 are shifted to larger values, but the majority is shifted into the other direction. These findings suggest that a more detailed investigation is necessary to understand how the attacks modify the inputs, possibly related to the different flavours.

4.1.2 Understanding the flavour dependence of adversarial attacks

To investigate how physical observables are distorted by systematic adversarial attacks (and Gaussian noise), selected features are visualized with additional histograms that are split by the flavour of the jet. A study of the flavour dependence can contribute to a deeper understanding of the working principle of the attacks, but it also helps deciding whether the currently used attacks are applicable or if the distortions become unphysical. These tests will be continued in Section 4.3.3 for the adversarial training, using the FGSM attack. Here, inputs are first distorted by a Gaussian noise term, then, the basic training is used in conjuction with an FGSM attack to obtain actual adversarial samples. The two approaches will be compared afterwards. It should be noted that all checks of the flavour dependence are carried out with relatively large parameters of the distortion ($\sigma = 0.02$, $\epsilon = 0.02$) to increase the visibility of potentially different effects. Distortions of that magnitude would be excluded by commissioning [58, 59] and only serve an illustrative purpose, most of the tests that follow afterwards when studying the performance and also the adversarial training mainly utilize smaller parameters.

The first feature used for the investigation of the flavour dependency, the signed impact parameter significance (2D) of the first track above the c quark mass threshold is shown in Fig. 4.3, where two histograms display raw and distorted inputs. This observable offers discriminating

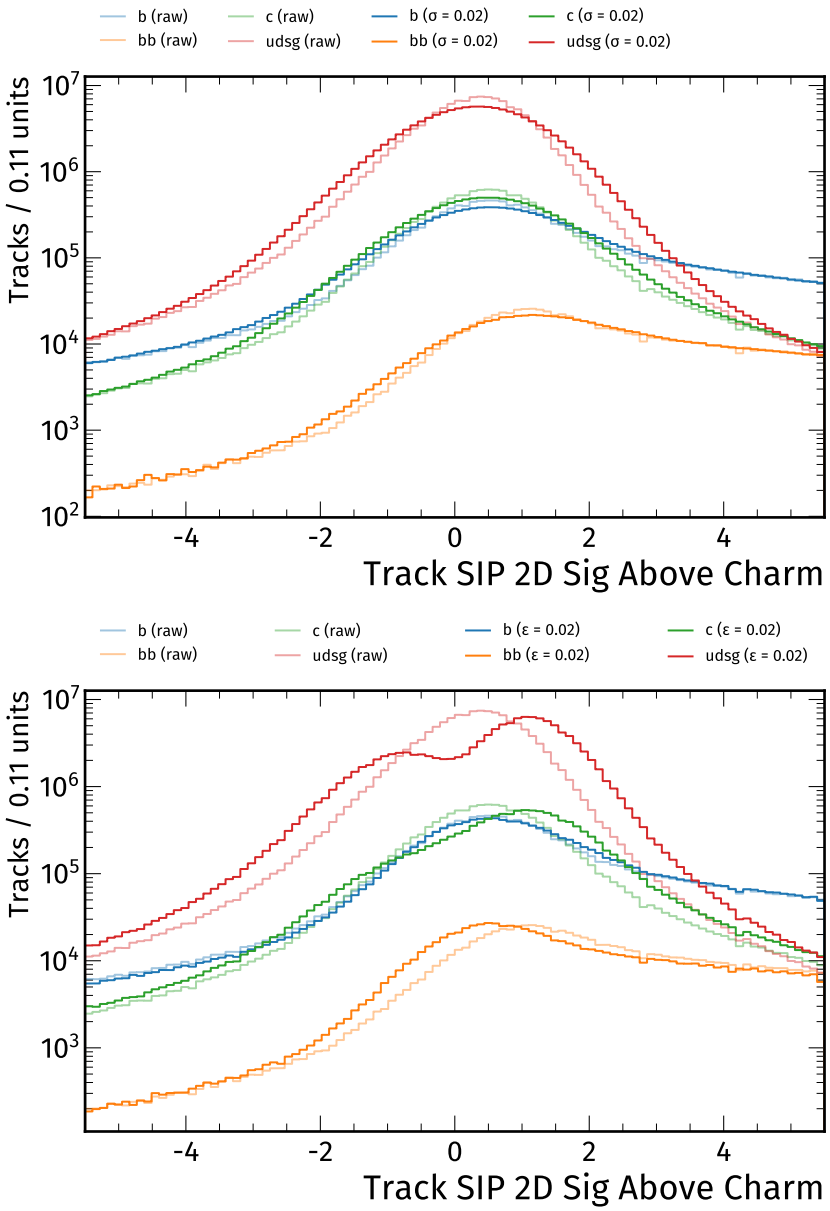

Figure 4.3: Detailed examination of the flavour-dependent impact of adversarial attacks, with a trend towards unphysical shapes. In this example, the 2D signed impact parameter significance for the first track above the charm mass threshold is depicted for raw and distorted inputs, with light and saturated colours, respectively. In a limited interval, using unusually large parameters, the distorted inputs in the top figure contain a noise term ($\sigma = 0.02$), while for the second figure, the FGSM attack has been applied ($\epsilon = 0.02$).

power by the fact that heavy-flavour jets are originally more abundant in the positive region of the distribution, whereas the distribution for light jets should be approximately symmetric [8]. The reason for this property can be found in the presence of a secondary vertex with displaced tracks for heavy-flavour jets, which is not the case for light-flavour jets. For those jets initiated by u, d or s quarks (and gluons), there is no preference whether the angle between the impact parameter vector and the jet axis is smaller than $\pi/2$ (”plus sign”), or outside this range (”minus sign”), as the decay happens close to the primary vertex (short lifetime of light hadrons) [8]. To some extent, this can be seen in the distributions for raw inputs, where the light red histogram is almost symmetric around zero, modulus some skewness due to long-lived

hadrons misreconstructed as jets or wrong clustering of tracks from heavy-flavour hadrons into light-flavour jets [8]. The (light) blue and orange histograms, which display the b and bb jets, have a long tail towards the positive side. For the c jets in light green, there is a less pronounced tail to the right, positive side. These findings match the expectations for undistorted inputs.

With the addition of Gaussian noise and related to the central limit theorem [108], the randomly distorted input shapes are spread out and tend toward a normal distribution, as can be seen in the top part of Fig. 4.3. Due to the quotient between the displayed range and the range originally filled by the distribution (see also Section 4.1.3), this effect is more pronounced for the 3D variant of the signed impact parameter significance (Fig. 4.4). There, the chosen parameter for random noise is definitely too high, leaving not much more than the information of a different mean value of the new distributions for the discrimination between different flavours. With Gaussian noise, the dip at zero vanishes and the distribution resembles a normal distribution [108]. When studying the performance of the model, it will be demonstrated if and by how much such random distortions influence the discrimination power. The neccesity to control the magnitude of the distortion will be discussed in Section 4.1.3. When the ad-

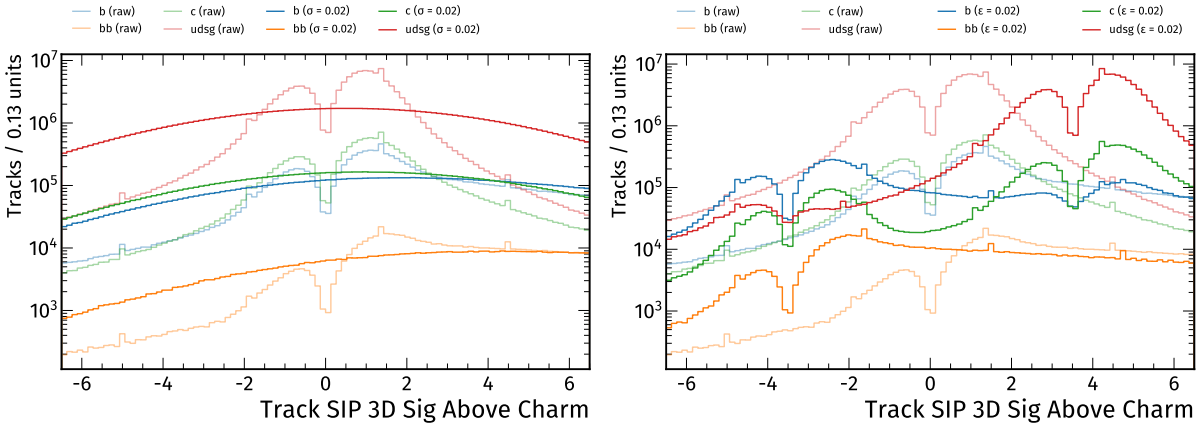

Figure 4.4: Detailed examination of the flavour-dependent impact of adversarial attacks, revealing dramatic unphysical effects. In this example, the 3D signed impact parameter significance for the first track above the charm mass threshold is depicted for raw and distorted inputs, with light and saturated colours, respectively. In a limited interval, using unusually large parameters, the distorted inputs in the left figure contain a noise term ($\sigma = 0.02$), while for the right figure, the FGSM attack has been applied ($\epsilon = 0.02$).

versarial attack is applied, the overall appearance of the input shapes seems to be translated mostly to one preferred direction per flavour, e.g. the adversarial inputs for bb jets appear more to the negative end, while light jets in the second row of Fig. 4.3 now obtain a double-peak structure of which the more abundant peak is shifted to the positive side (in Fig. 4.4 (right), the conclusion is similar, but with a quadruple-peak structure). For c jets, there is a slight trend to a double-peak structure similar to the one for light-flavour jets, the distribution for b jets receives practically no change (see the lower part of Fig. 4.3). This could be a hint that the given variable is a driving factor only for the identification of a certain subset of all flavours, e.g. for light or charm jets, while the discrimination power of the full 3D variable is sensitive to changes of any flavour. Investigations regarding the feature importance have been started and should lead to better insights when comparing for example the AUC-ranking of every feature before and after an attack. In general, the attack mostly "inverts physics", or in other words, it destroys the expected behaviour for different flavours. The resulting distributions are unphysical, although the attack does not have the knowledge of how to create unphysical results and no step in the creation of the adversarial examples explicitly tells the

attack to produce opposite-to-normal features. An FGSM attack is able to figure out on its own which direction is most promising for a systematic first-order distortion [64], a confirmation that the construction based on the loss function does indeed work. As will be discussed later, the benefit of easy to generate adversarial examples plays a subdominate role, but it is important that all inputs are only shifted according to the allowed boundaries, limited by the systematic uncertainties [4, 109]. Another way to describe this necessity is that the distortions should stay "invisible", or that raw and distorted histograms show compatible distributions. More complex manipulations that stay undetectable and imitate simulation mismodellings, potentially by introducing or adjusting correlations between features while decreasing the model performance, are what to look for with future investigations [4, 63]. The physical conditions should not be completely compromised by adversarial attacks and constraints need to be followed.

4.1.3 Applicability of the FGSM attack in its current form

The FGSM attack as implemented for the present studies does provide a systematic distortion of the inputs, however, to exclude unphysical behaviour, improvements of the setup are necessary. In the following, the current status of the implementation as well as the corresponding limitations that demand further refinements are reviewed, and some possible strategies to adjust the setup are proposed. Although this section focuses on true adversarial attacks, the constraints discussed below also apply to the generation of noise terms, unless stated otherwise.

Handling of default values. Compared to Ref. [9], default values now only occur at one given position for every input distribution, and the rescaling into the original input range places the default bin at the same position as prior to any scaling, outside of the main part of the distribution, for raw and distorted inputs. This means that default values are currently not varied by any attack (also no "slight" modifications), because a single quantity can not be "partially" defaulted. Also, default values are correlated between several features, e.g. all features that contain information for tracks are ordered in the same way [8], and if one per-track variable would switch its defaulted-status, there needs to be a definition what should happen to related quantities, subsequently. After all, a change of existing default values is unphysical, as a value that contains no meaningful information can not suddenly receive new information. A more natural distortion would introduce additional defaults to previously valid values, but then, not to replicate theoretical mismodellings in simulation, but rather to model malfunctions of the detector components or errors during the reconstruction or data taking [4, 8]. The whole problem complex of default values could be addressed by moving to taggers that utilize more low-level information without imposing additional (*a priori*) selection criteria [9, 8, 51]. From a technical point of view, instead of comparing each scaled default input with the corresponding original value and allowing an error margin due to floating point precision, a safer way would be to store a boolean that contains the information whether some quantity has a valid (0) or a default value (1).

Handling of integer variables. Similar to default values, also integer variables are currently not modified specifically, as pointed out in Ref. [9]. The restriction has to be made because a slight change of an integer quantity does not result in whole numbers, therefore leaving or exceeding the allowed feature space. Variables that are excluded from adversarial attacks (and the application of noise) are all variables that count something: the number of secondary vertices in the jet, the number of selected tracks, the number of track for which η_{rel} is computed, the number of tracks associated with the secondary vertex; and the variable that encodes the vertex category (see Tables 2.2, 2.3 and 2.4, as well as Ref. [8]). Currently, in the available data

sets there is no access to the low-level inputs from which high-level observables are obtained. Therefore it is impossible to replicate any simulation artefacts related to integer variables that for example count a number of selected tracks or vertices or represent the category to which a secondary vertex belongs to (if it exists). These features are directly related to the reconstruction step and it would be unphysical to change them *a posteriori* by an adversarial attack. Again, these quantities are connected to the track and secondary vertex properties obtained after imposing certain selection criteria [8]. Any method that aims at modifying the integer variables must pass on additional information to all correlated features. An example would be the number of selected tracks variable: if a distortion decreases this quantity for a given jet, all track variables that require this number of tracks (or more) need to be defaulted. Possible strategies to treat integer values nonetheless involve fitting a function to a histogram, shifting that continuous distribution, and iteratively placing inputs in a new histogram that reflects the shifted function in discrete steps [92]. This would need to work with large numbers of inputs, in a reasonable time, which is subject to future developments.

Limiting the upper bounds of the distortion per variable to incorporate physical constraints and to comply with uncertainties of measured observables. The current implementation of the FGSM attack (as well as the addition of a noise term) utilizes the same limiting parameter (ϵ or σ) for all features. For scaled quantities (mean at zero, standard deviation of one) this would be a valid technique to handle all inputs equally. But the actual application of the attacks will be visualized for different types of physical observables with their original units and with a rescaled x axis. If the displayed range and the totally filled range from which the scaled inputs are derived do not match, a given distortion of fixed size ϵ (σ) will show a different impact for different distributions. A first solution to this problem could be to multiply the parameters of the distortion with a specific factor per feature that includes the scaling, obtained by comparing the original ranges with the interesting ranges, taking care of the varying width of the distributions. In order to do this systematically, the attack needs to know the range in which all features will be plotted and up to which refined binning the distortions should stay undetectable. Related to this necessity, all distortions would need to be compared with commissioned plots of the given input variables to compare the variations to the (systematic) uncertainties assigned to the different features [58, 59]. Without using methods from commissioning, a quick strategy could implement a check that computes a distance measure between two distributions, like the Kullback-Leibler (KL) divergence [110], to find a suiting parameter that ensures the distorted and raw shapes do not differ too much. This measure is defined in another context in Eq. (A.1) in the appendix and showcases the loss of information content between two distributions [110, 15]. Once the interesting parts of all distributions and the uncertainties are known, an already existing implementation could be extended for an iterative application of the KL divergence while creating adversarial samples of increasing magnitude, up to some threshold criterion. Inherent features of the different distributions would need to be included for the definition of meaningful upper bounds, for example a distribution that is flat (and smooth) will show smaller impacts of an attack than a distribution where already for raw quantities, large fluctuations or prominent peak structures appear. Also the flavour dependence and discriminating power of the different features would need to be considered to derive upper bounds for the distortion, because the response to the same type and severity of an FGSM attack might vary for different flavours. This is related to the fact that the loss function depends not only on the inputs, weights and bias terms, but also on the targets. All flavours therefore carry their own "loss manifolds", which can have completely different properties regarding the convexity and the possible impact of an adversarial attack [5, 62, 65]. On another note, Ref. [111] mentions that regular, unconstrained adversarial examples leave the underlying data manifold of a given class ("off-manifold"), which can be related to the unphysical shapes. With "on-manifold" adversarial samples that

are still within the true decision boundary, but outside the classifier's learned decision boundary, the generalization through adversarial training could be more effective [111].

Expected impact of the current attack, future extensions and relation to mismodellings in simulation. As explained above, the creation of adversarial inputs does not encompass the full feature space that is available for the training. Whenever the impact of this pruned approach is discussed, the following property has to be taken into account: the changes in performance as studied in the following sections could be larger if all variables would be varied by the attack (including the five integer variables) and if defaults would contribute as well (roughly two thirds of the jets have default values, mainly driven by the secondary vertex (and track) variables, see Section 2.2.4). The following investigations can not give an upper bound for the susceptibility towards arbitrary adversarial attacks, all conclusions are only applicable to first order, because the FGSM attack is a first-order adversarial attack [64], which furthermore does not take all inputs into account. If there are mismodellings in the simulation that are of the kind introduced above ("on-manifold" in the data space, but possibly resulting in disjoint areas of the loss surface [111]), these can not be modelled with the current type of attacks. In that case, the investigations might overestimate the robustness [111, 56, 64]. It is also not clear if the adversarial attack produces out-of-distribution samples [56], or not. Therefore, neither an upper- nor a lower-bound for the susceptibility of the network can be stated. These limitations will be addressed again in Chapter 5, but the following investigations are carried out based on the restricted attacks.

4.2 Performance on distorted inputs

With the help of the previously introduced metrics (see Chapter 3), the performance of the basic model will be compared between raw and distorted inputs. Similar investigations that take the adversarially trained model into account follow in Section 4.3.

4.2.1 ROC curves

As discussed earlier, this thesis focuses on b-tagging, therefore the following ROC curves are obtained exclusively for the BvsL discriminators. One should expect that the qualitative behaviour does not differ substantially between the different flavours, although a quantification of the different impacts would need to follow. All investigations include the full pseudorapidity and transverse momentum range that is available after the cleaning, but it is possible that restricting the tests to certain small intervals of η and p_{T} shows varying impacts of the attacks.

Fixed epoch, different parameters for the distortion (FGSM attack)

For this comparison, the basic model is evaluated at epoch 200 (the last epoch of the training), inserting raw and distorted inputs. Figure 4.5 includes BvsL–ROC curves with varying parameters ϵ. As expected, the model performs best on undisturbed test samples, reaching an AUC of 0.9473 that has been found to be almost identical to that of DeepCSV (see Fig. 3.5), while increasing the parameter for the distortion reduces the performance accordingly. Moderate choices like $\epsilon = 0.005$ or $\epsilon = 0.01$ are enough to cause a signifcant loss of performance, where the AUC decreases by several percent to 0.9236 and 0.8942. As discussed above, physical constraints restrict the possible parameters for the adversarial attack, which means that the curves for even larger distortions are just meant to be illustrative. Such large discrepancies are not expected, at least not from first-order adversaries. Another hint that the red (and violet) ROC curves are too extreme and that they overestimate the expectable susceptibility

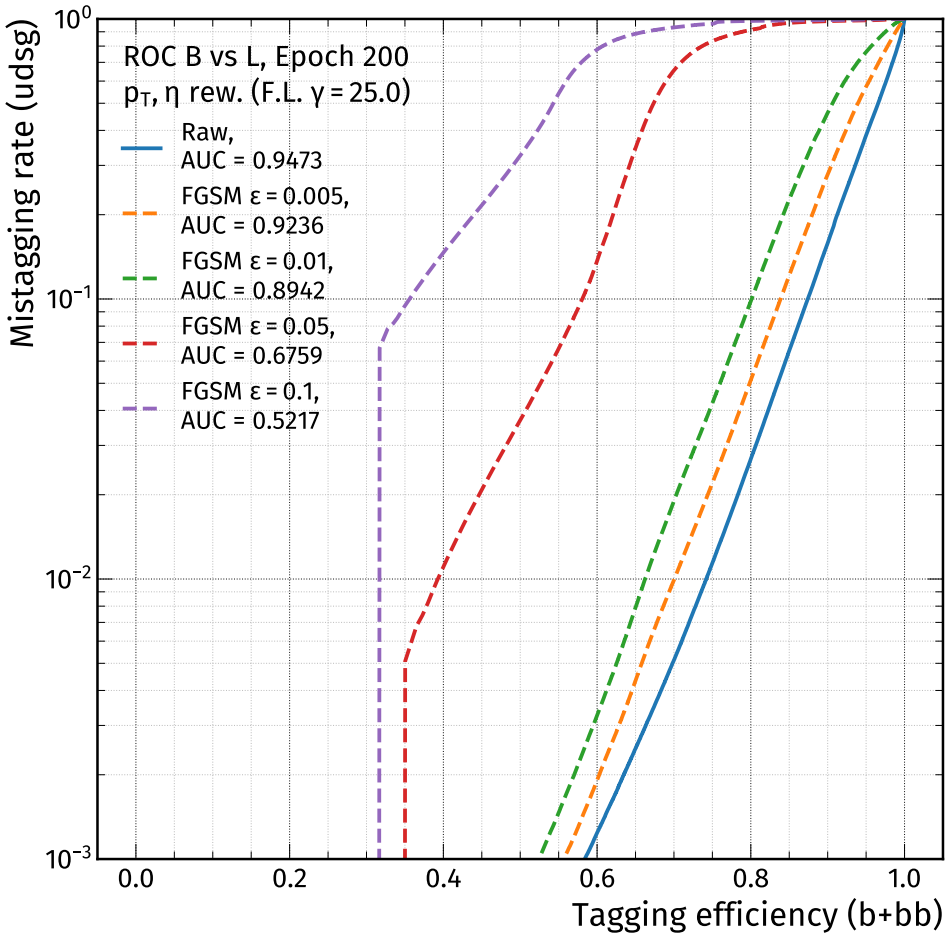

Figure 4.5: ROC curves for the BvsL discriminator, evaluated at epoch 200 with raw and adversarial inputs, using different parameters ϵ for the FGSM attack on the baseline model.

is the cutoff at tagging efficiencies of 0.35 (0.31). This indicates that the discriminator distribution abruptly ends and that it does not contain entries below a certain non-zero threshold, an improbable behaviour for possible simulation artefacts when comparing the distribution to data during commissioning [8, 58, 59].

Fixed parameter for the distortion, different epochs (FGSM attack)

This next evaluation keeps the parameter for the distortion fixed, but while comparing raw and adversarial samples, the training epoch is varied. For Fig. 4.6, epochs 10, 100 and 200 have been chosen. With this setup, the changes in susceptibility of the model during the training can be investigated, excluding the earliest iterations through the whole data set where large fluctuations in performance might appear. All FGSM attacks use $\epsilon = 0.01$ for this test. This check shows that the model becomes more and more susceptible to the same type and absolute impact of adversarial attacks, the longer it is trained. All solid lines show the evolution of performance on raw data, while the dashed lines (in corresponding colours for each epoch) demonstrate the performance on systematically distorted inputs. While the performance on raw inputs increases with the number of epochs (note how the blue, orange and green solid lines move to the lower right), the dashed lines that depict the performance on FGSM inputs are reversely ordered. There, the blue, orange and green dashed ROC curves move to the upper left, where the performance decreases with more epochs. The impact of the FGSM attack is lowest for epoch 10 (blue), where the AUC decreases from 0.9424 to 0.9205. On the contrary,

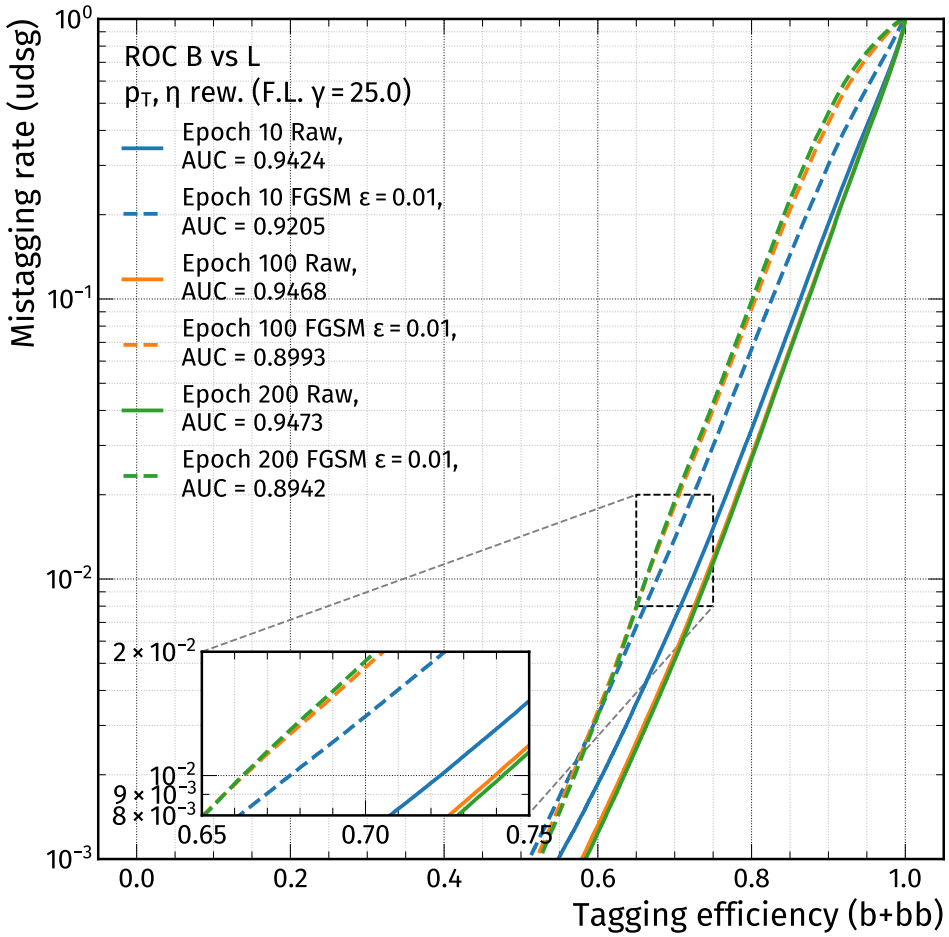

Figure 4.6: ROC curves for the BvsL discriminator, evaluated at epochs 10, 100 and 200 with raw and adversarial inputs, using $\epsilon = 0.01$ for the FGSM attack on the baseline model, crafted for each epoch individually.

the epoch with the highest raw performance (epoch 200 in green) is the least robust out of the three checkpoints shown in Fig. 4.6 (the AUC drops from 0.9473 to 0.8942). Although the raw performance measures differ by less than a percent, the systematically distorted performances differ by more than 0.02 in AUC between epochs 10 and 200. This leads to the conclusion that there is a tradeoff between model performance and robustness [6, 5, 63], a property that will be revisited in the following section by following the evolution of the area under the ROC curve.

ROC curves with randomly distorted inputs (Gaussian noise term)

The following Fig. 4.7 contains similar checks as before, but instead of using a systematic FGSM attack, the inputs are now varied randomly by adding a Gaussian noise term. As expected, the impact on the performance is much smaller compared to the tests with the FGSM attack, both when looking at different parameters in the left subfigure, as well as when probing the dependence of the vulnerability on the epoch, as shown on the right side. Even with a drastic change of the inputs by manipulations of the order $\sigma = 0.1$, the model keeps an AUC of 0.9113 at epoch 200. The comparison of different epochs shows a tiny separation of the green curves (epoch 200, raw and distorted), while the blue curves (epoch 10) overlap entirely for the ranges of efficiency and mistagging rate displayed in the figure. This indicates that noise does have a marginal impact on the performance, where the tradeoff between nom-

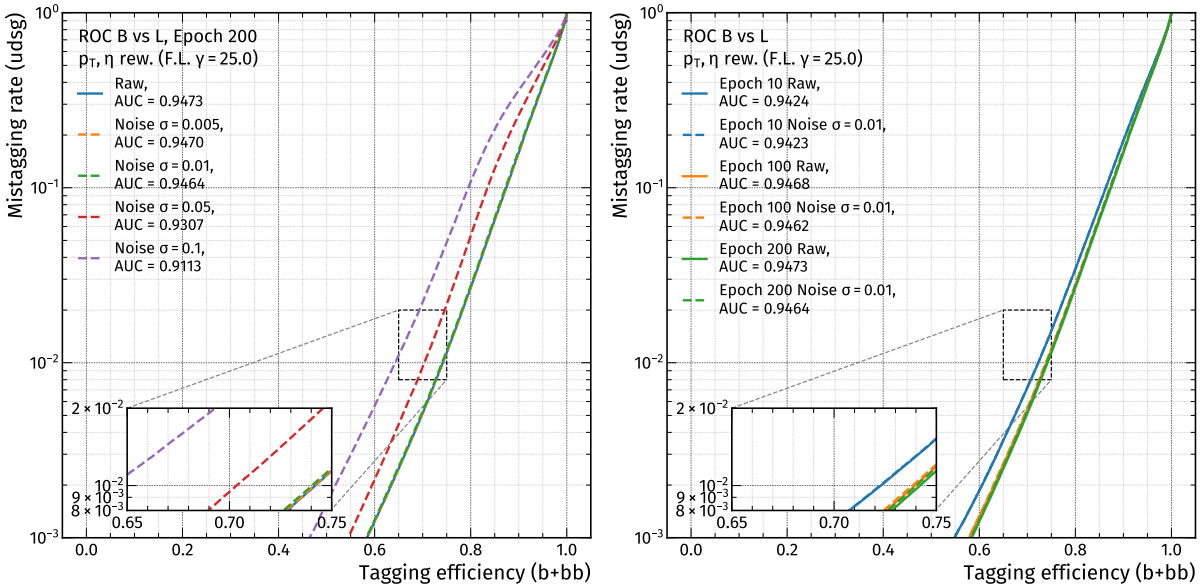

Figure 4.7: ROC curves for the BvsL discriminator, evaluating the basic model with raw and randomly distorted inputs (Gaussian noise), either with a fixed epoch with different parameters σ, or with a fixed parameter σ, but evaluated at different checkpoints of the training. The left subfigure uses epoch 200, different colours represent different magnitudes of the distortion. On the right, a fixed parameter $\sigma = 0.01$ (dashed lines) is utilized in comparison with raw inputs (solid lines), for different epochs in different colours.

inal performance and robustness is not as exaggerated as for the systematic adversarial attack (FGSM). The difference in AUC for raw and distorted inputs increases from around 0.0001 to 0.001 between epochs 10 and 200, by a factor of ten. However, the absolute scale of this impact is so small that systematic uncertainties would cover this effect [8, 48] also for the latest checkpoint of the model that is expected to be most sensitive to mismodellings [4, 5, 64].

4.2.2 AUC over epoch

While training the model, a checkpoint with the current model parameters is stored for every epoch with which the inference on test data can be done. One interesting quantity to monitor during the training phase is the AUC, as introduced in previous sections. With the help of raw and distorted inputs, the discrimination between b+bb and light jets will be investigated. The resulting visualizations (see Fig. 4.8) show the impact of a Gaussian noise term as well as an FGSM attack with varying parameters on the absolute value of the AUC. Additionally, the ratios between the raw and disturbed AUC values are displayed in Fig. 4.9. In this section, all checks only cover the baseline classifier (without adversarial training).

Absolute values (Gaussian noise and FGSM attack)

The blue curves for both subfigures of Fig. 4.8 are identical, they show the evolution of raw AUC during the training of the basic model between epoch 1 and 200. As can be seen, the improvement in performance is only marginal and changes between epochs become smaller, the longer the model is trained. It would be too early to conclude that the model has converged, as some residual slope is visible in the additional panel that covers a smaller range in epoch and AUC. One difference between both subfigures (noise on the left, FGSM on the right) is the absolute impact of the disturbance on the AUC while comparing the same magnitudes of the

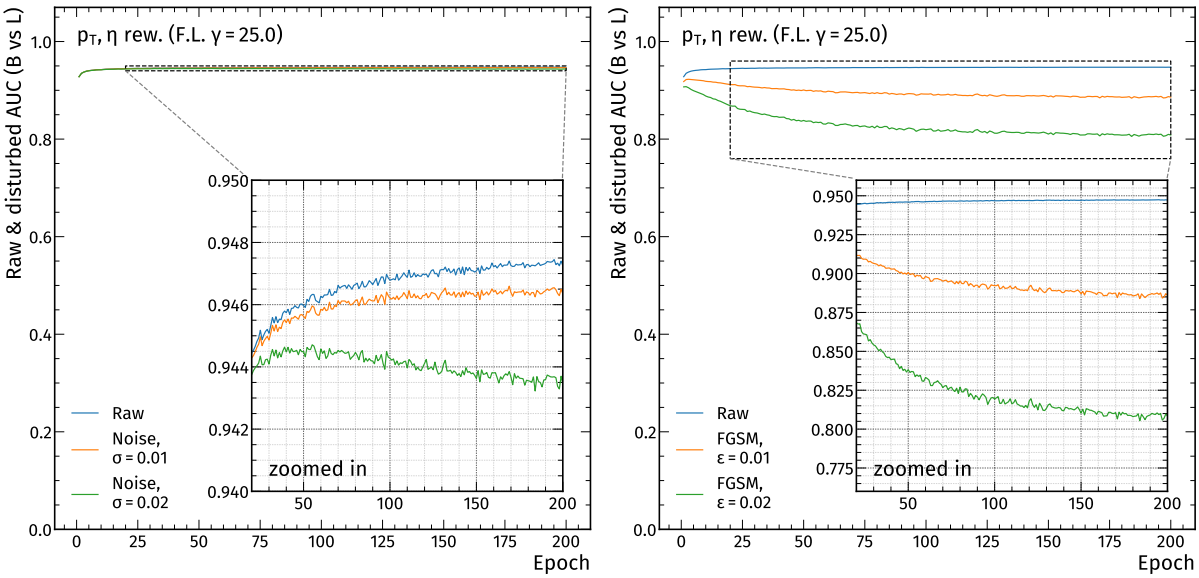

Figure 4.8: Evolution of the BvsL–AUC with the number of epochs for the basic training, evaluated on raw and distorted inputs (adding noise with $\sigma = 0.01$ and $\sigma = 0.02$ (left) or by using the FGSM attack with $\epsilon = 0.01$ and $\epsilon = 0.02$ (right)).

parameters (e.g. comparing $\sigma = 0.01 = \epsilon$ (orange) or $\sigma = 0.02 = \epsilon$ (green)). The scale between noise and FGSM had to be adjusted, because random disturbances reduce the AUC significantly less than the systematic approach (FGSM). To give an example, even at epoch 200 the difference between raw and noise performance amounts to about $0.1\,\%$ for $\sigma = 0.01$, while the FGSM attack (with $\epsilon = 0.01$) reduces the performance at that same epoch by roughly $5\,\%$. This feature is apparent also for earlier epochs. For both setups (noise / FGSM), the impact becomes larger as the number of epochs is increased, this highlights the tradeoff between (raw) performance and robustness of the model. The functional behaviour differs slightly between random and systematic distortions, the point after which the performance on distorted inputs does not increase anymore occurs at a later epoch for Gaussian noise than for the FGSM attack. For example, the green curve for random distortion with $\sigma = 0.02$ starts to fall at about epoch 50, up to which the reduction in performance with respect to raw inputs is only given by an almost constant offset. This point of peak performance on distorted inputs for the FGSM attack appears at one of the earliest epochs, when the number of iterations through the whole training set is about 2. These findings imply that the functional dependence of the susceptibility on the number of epochs differs between noise and FGSM and that the properties related to the first (and second) derivative of the distorted AUC-over-epoch curves develop later for noise than for the FGSM attack.

Ratios (Gaussian noise and FGSM attack)

The ratios between raw and distorted AUC, evaluated at epochs 1 to 200 for the basic training, are shown in Fig. 4.9, again with noise on the left, and FGSM on the right. The previous paragraph already introduced the slightly different functional dependence for the AUC-over-epoch relation between noise and FGSM, also the ratios demonstrate the different effects. The ratios obtained when adding Gaussian noise could very well represent a clipped part of the ratios that are visible for the earliest epochs when comparing raw with FGSM inputs. Other than that, the ratios confirm the different scale of the impact between noise and FGSM.

As the comparisons have shown, the basic model is particularly susceptible to (first-order) adversarial attacks, while it also shows some vulnerability towards randomly distorted in-

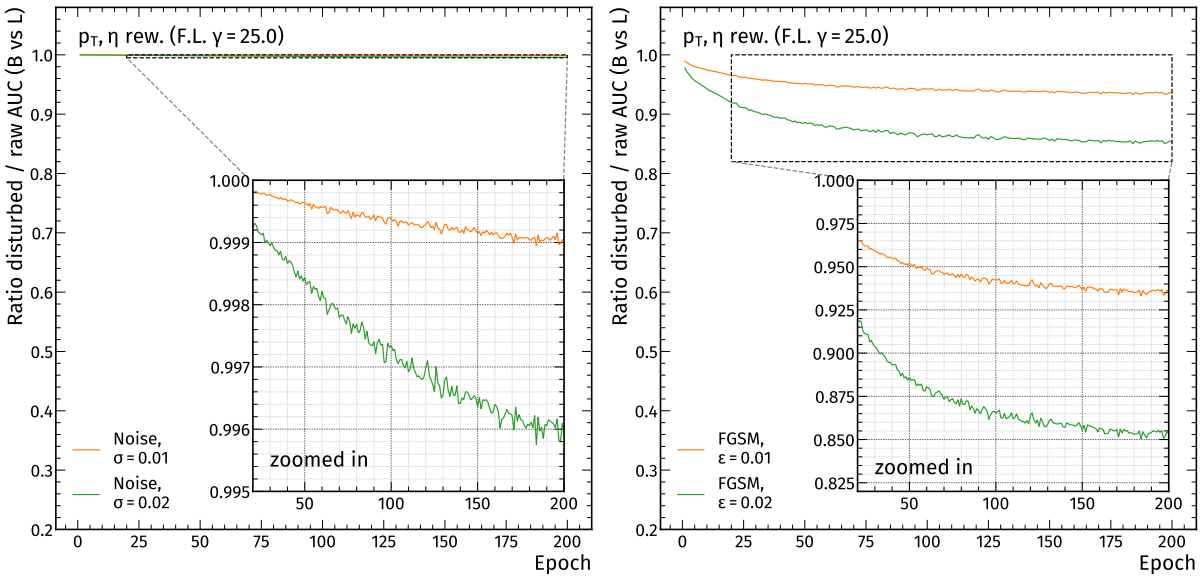

Figure 4.9: Evolution of the ratio between distorted and raw BvsL–AUC with the number of epochs for the basic training, evaluated on raw and distorted inputs (adding noise with $\sigma = 0.01$ and $\sigma = 0.02$ (left) or by using the FGSM attack with $\epsilon = 0.01$ and $\epsilon = 0.02$ (right)).

puts. This leads to the next section, where an attempt is made to improve the robustness of the network.

4.3 Influence of adversarial training

Adversarial training and the motivation behind this technique have been introduced in Section 1.6.3. The model itself is not modified entirely, the only difference to the basic training as explained previously is that the adversarially trained model only learns from distorted inputs.

4.3.1 ROC curves (FGSM attack)

To evaluate how the adversarial training influences the model performance on raw and distorted inputs, this section demonstrates BvsL–ROC curves in the style utilized for the previous studies with the basic training. In Fig. 4.10, either the epoch is fixed (left subfigure) or the tests use a fixed parameter for the distortion (right subfigure) with the FGSM attack. In contrast to Fig. 4.5, where different magnitudes of the distortion are compared for the basic training at epoch 200, such distortions show a smaller impact for the adversarial training (also at epoch 200, as shown on the left side of Fig. 4.10). This is evident for all parameters probed, additionally, the cutoffs for large parameters of ϵ appear at smaller efficiencies and mistagging rates for the adversarial training. For example, the cutoff for $\epsilon = 0.05$ is now not visible anymore. Overall, without quoting the exact AUCs, the behaviour that is observed for the basic training is repeated also with the adversarial training: larger distortions of the inputs lead to smaller values of AUC with which the performance is measured. The right side of Fig. 4.10 uses only one fixed parameter ($\epsilon = 0.01$) for the FGSM attack, exactly the same magnitude that is applied during the adversarial training. The different colours represent different epochs, with solid lines for raw inputs and dashed lines for distorted inputs. This means that the setup for the comparison is exactly the same as for the comparison carried out with the basic training in a previous section. The conclusion however is fundamentally different: with the adversarial training, no increase of the susceptibility toward the adversarial attack with the number of

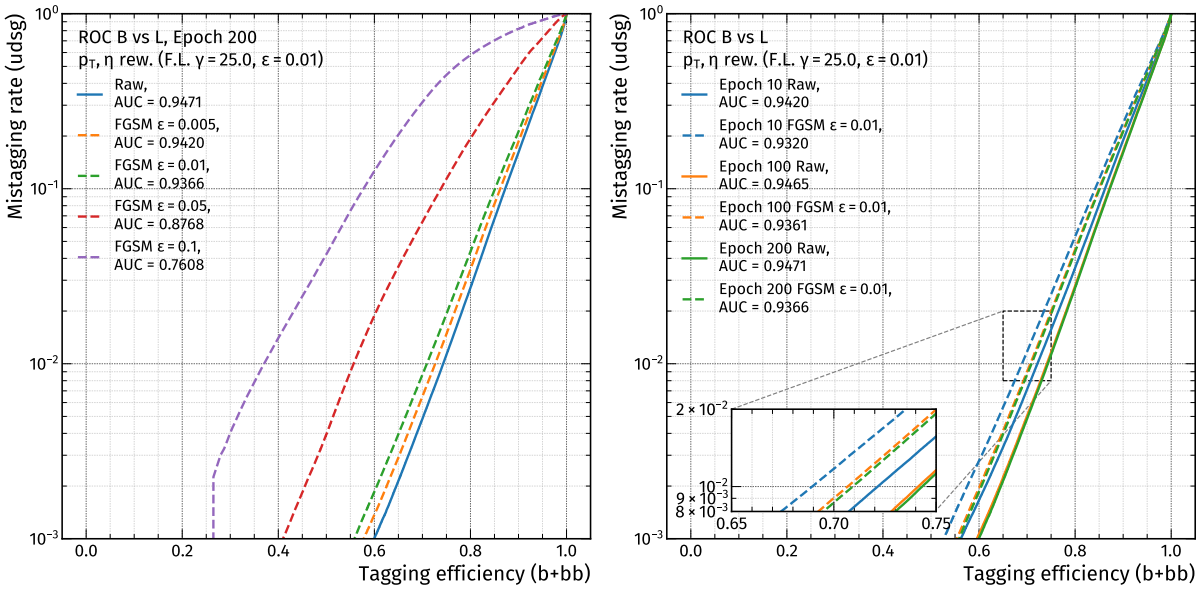

Figure 4.10: ROC curves for the BvsL discriminator, evaluating the adversarially trained model with raw and distorted inputs (FGSM), either with a fixed epoch with different parameters ϵ, or with a fixed parameter ϵ, but evaluated at different checkpoints of the training.

epochs is directly visible. The order of all visualized distorted ROC curves matches the order of the ROC curves for raw inputs (blue, orange, green, which means that the curves for epochs 10, 100, 200 move to the lower right for the raw (solid line) and distorted (dashed line) case). This does not mean that the adversarially trained model is completely robust to first-order adversarial attacks, indeed some susceptibility is still there, but the tradeoff between performance and susceptibility seems to be reduced. For the three epochs compared so far, more training (that means more iterations through all training samples) does not increase the vulnerability of the adversarially trained model. A more detailed investigation of the behaviour with respect to the number of epochs follows when evaluating the BvsL–AUC at all checkpoints later in this section.

The next comparison shows the basic and adversarially trained model in Fig. 4.11 simultaneously (using the colours blue and orange, respectively). The setup is similar, there are unseen test samples with which BvsL–ROC curves are created and the evaluation either uses the raw inputs or distorted inputs that are obtained from a FGSM attack with $\epsilon = 0.01$ (individually per model). Both trainings are investigated at epoch 200. The first noticeable property is that the basic and adversarial training offer an almost identical performance on (the same) unseen, undisturbed test inputs. This is remarkable, because the adversarial training only uses distorted inputs during the training, where the model never "sees" any raw inputs at all. Both solid lines overlap (evaluation on raw inputs), which means that both models achieve approximately equal nominal performance. The raw AUC is 0.9473 and 0.9471 for the basic and adversarial training, respectively. This is a very small difference and in the ROC curves, tiny deviations occur at low mistagging rates (adversarial training is better), while the basic training achieves marginally better performance at mistagging rates above the medium working point, almost too small to be seen. The performance on distorted inputs (FGSM) instead shows a large discrepancy between the basic and adversarial training: over the full displayed range in the efficiency–mistagging rate plane, the adversarial training outperforms the basic training (dashed lines). This means that the adversarial training is less susceptible to first-order adversarial attacks, namely the type of attack that is applied during its training phase [64]. At the same time, the nominal performance is not compromised by this feature of the adversarial training, which surpasses the expectations (see [63, 5]). The improved robustness

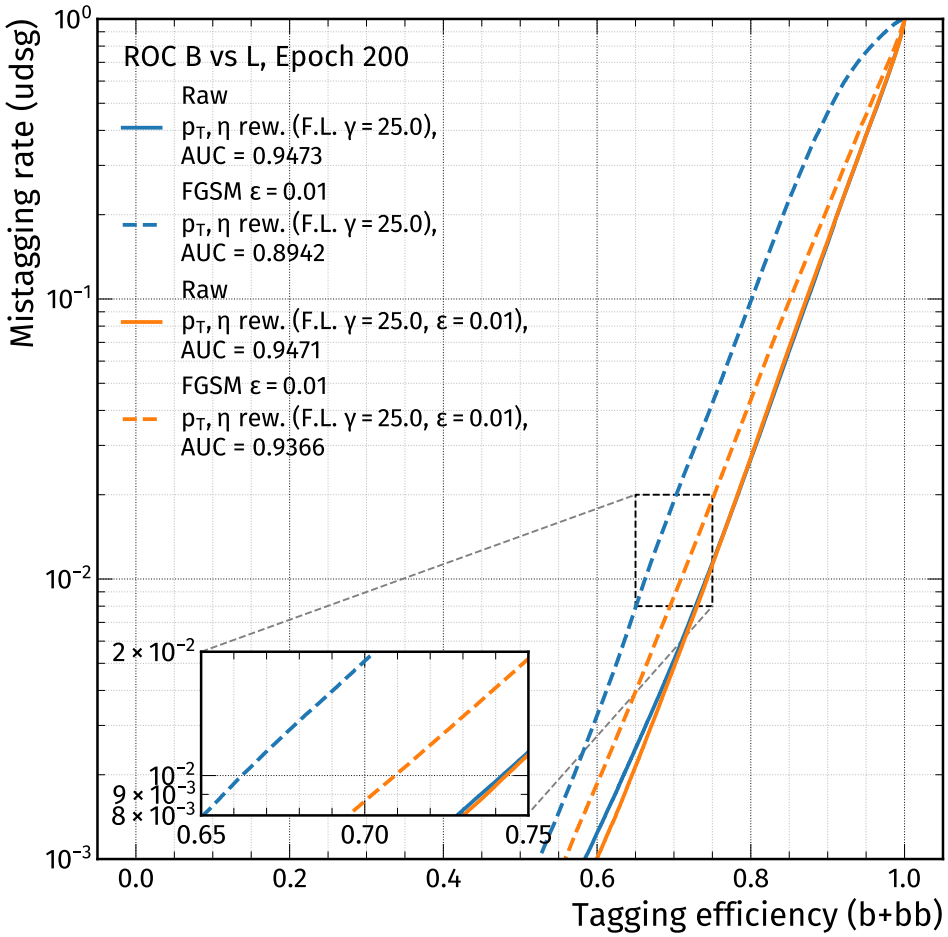

Figure 4.11: ROC curves for the BvsL discriminator, evaluated at epoch 200 with raw and adversarial inputs, using $\epsilon = 0.01$ for the FGSM attack applied individually to the basic model as well as the adversarially trained model. Raw performance is visualized with solid lines, the performance on distorted inputs uses dashed lines. The basic (adversarial) training is shown in blue (orange), respectively.

to distorted inputs therefore is not only a pleasant side effect of a model that has learned to do well only on distorted inputs by exactly memorizing the mapping from (distorted) inputs to the truth labels. Instead, the model does indeed generalize better to inputs inside the ϵ-ball around the original inputs. At least in the direct vicinity of the raw inputs and to first order, the FGSM attack yields the worst-case distortions of the inputs, individually for both training setups. To show that also the injection of the same adversarial inputs confirms the higher robustness of the adversarially trained model, Section 4.4.2 provides another test. This is done to rule out that the adversarial attack on the adversarially trained model is slightly biased by the fact that it is also used during the training step, potentially overestimating the robustness.

4.3.2 AUC over epoch

Also for the adversarially trained model, the training is checkpointed after every epoch to allow detailed evaluations, like the investigation of AUC-over-epoch for the BvsL discriminator as shown in Fig. 4.12, using Gaussian noise (left) or the FGSM attack (right subfigure). As can be seen from the additional panel, when adding random distortions the change in AUC is only marginal. With the moderate $\sigma = 0.01$ (orange), the AUC-over-epoch curve overlaps with the blue curve for raw inputs, where fluctuations between epochs are larger than the

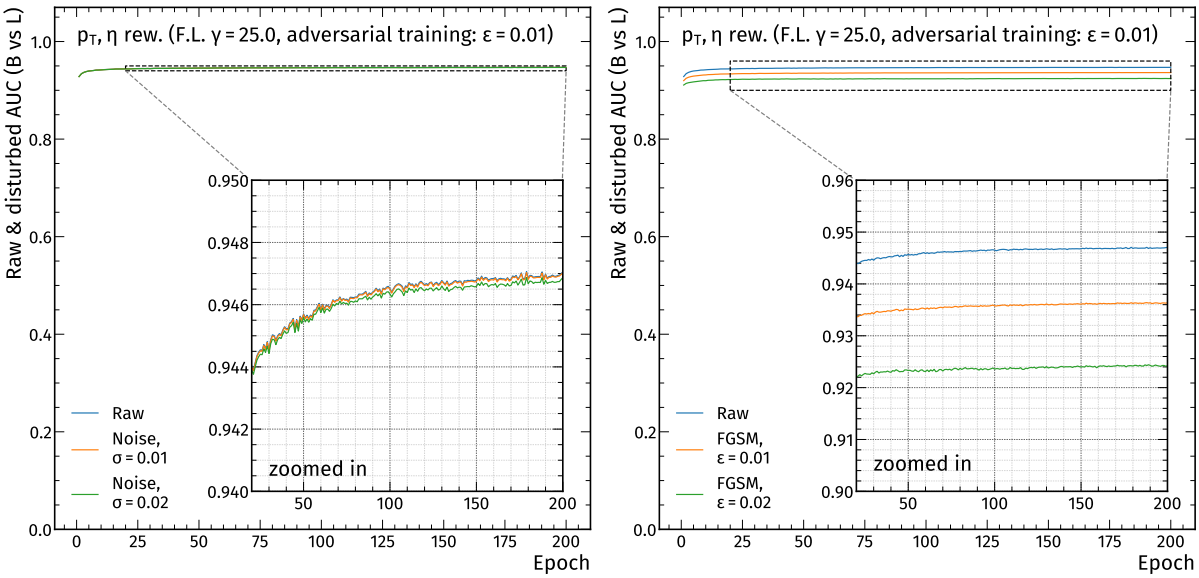

Figure 4.12: Evolution of BvsL–AUC with the number of epochs for the adversarially trained model, evaluated on raw and distorted inputs (adding noise with $\sigma = 0.01$ and $\sigma = 0.02$ (left) or by using the FGSM attack with $\epsilon = 0.01$ and $\epsilon = 0.02$ (right)).

differences between raw and distorted inputs. Even doubling the magnitude of the impact (green curve) only slightly shifts the displayed AUC values over epoch, maintaining the intrinsic fluctuations. Only a tiny trend towards slightly larger differences in AUC for more epochs is visible, when comparing this to the starting phase of the adversarial training (up to around epoch 50 the curves basically coincide). The subfigure on the right shows that also a systematic distortion of the inputs introduces no visible dependence on the number of epochs, for a given parameter ϵ, the differences between raw and distorted AUC stay approximately the same (ratios are constant). Hence, the FGSM attack only leads to a constant offset, but the tradeoff between model performance and susceptibility is practically gone. It should be noted that despite the fact that the attacks (as well as the addition of the noise term) are known to produce several input distributions that should be excluded by commissioning [58, 59], the adversarially trained model is able to keep the performance at a very high level, for all epochs investigated. Surely this does not exclude that more complex mismodellings in simulation could have a larger effect on the vulnerability, but at least for the first-order attacks studied in this thesis the adversarially trained model is a lot more robust than the model trained on raw inputs only [64]. More tests, also regarding possible correlations between inputs [4, 63], still need to follow.

4.3.3 An attempt to understand robustness and generalization capabilities of the adversarially trained model

In this section, the jet flavour dependence of the adversarial attacks will be used to build a possible explanation for the higher robustness of the adversarially trained model, compared to generic training on raw inputs only. Figure 4.13 shows the two quantities that have been studied previously for the basic training, but now before and after the application of an FGSM attack for the adversarial training. The description leads to a potential influence of the geometry of the loss surface [62, 65, 112, 113], which will be explained with the help of Fig. 4.14. When looking at the systematically distorted inputs, one might conclude that the adversarial attack for the adversarially trained model is almost like "coin-flipping". While the distorted shapes for the basic training show a prominent asymmetry (e.g. Fig. 4.3, second row), espe-

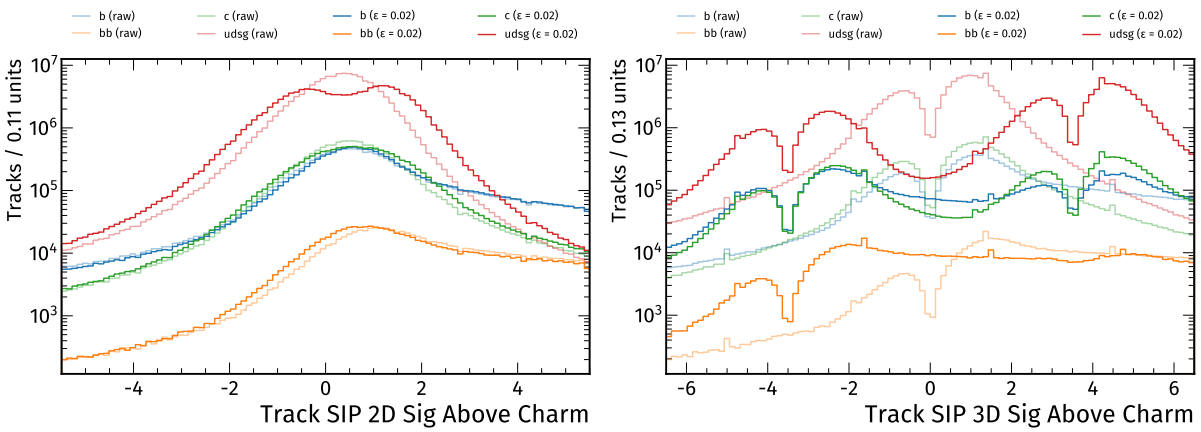

Figure 4.13: Applying the FGSM attack based on an adversarially trained model shows suppressed flavour-dependency and relatively symmetric shapes. The attack uses the parameter $\epsilon = 0.02$, which is higher than the moderately chosen parameter of $\epsilon = 0.01$ during the modified training loop. For comparison, the left figure demonstrates the impact for the 2D signed impact parameter significance for the first track that raises the invariant mass above the charm quark mass, while the right figure depicts the corresponding 3D observable.

cially for light jets (red histograms) the effect is more symmetric when using the adversarially trained model (see Fig. 4.13, both histograms). For the 3D signed impact parameter significance (first track above charm threshold), there is almost only a shifted double-peak structure (like a copy of the original distribution, shifted to the right for light jets when using the basic training), while the adversarial training leads to adversarial examples with a clear quadruple-peak structure. There, the attack has difficulties deciding which direction is the worse direction. The illustration in Fig. 4.14 suggests that the geometry of the loss surface

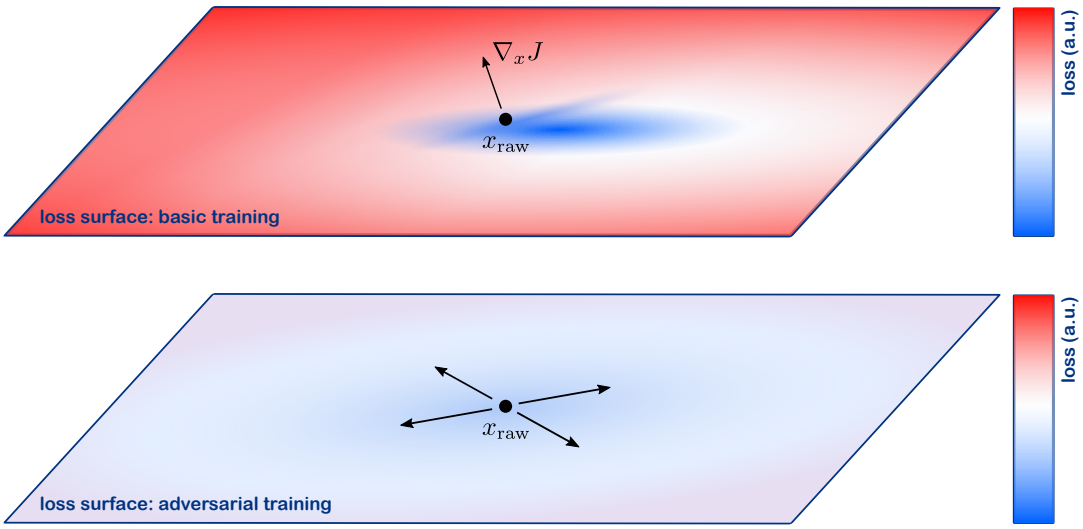

Figure 4.14: Illustration of the assumed geometry of the loss surfaces for the basic as well as the adversarial training. Inspired by Refs. [112, 65, 62]

can be particularly different between the basic and adversarial training. A feature that could explain the asymmetric shift of inputs for the basic training while producing more symmetric shapes for the adversarial training is the flatness of the loss surface in the vicinity of the raw inputs [112, 65, 62]. If the basic training only learns from raw inputs, it is conceivable that there will be a preferred direction for a subsequent gradient ascent that maximizes the loss function. For a given flavour, there will be a specific vector pointing away from a local min-

imum that could explain why the adversarial inputs generated for the basic training show a clear preference for the direction into which they are shifted. The adversarial training always sees (new) adversarial inputs, so the adjustment of the model's parameters might average out over epoch, always following the newly distorted inputs, to end up with a locally constant loss manifold around the original inputs. This would mean that not the exact memorization of training data, but rather correlations or more complex hidden structures contribute to the improvement of the performance for the adversarial training [64, 63, 65, 62]. With the assumption of a flat loss surface close to the raw inputs there would be no preferred direction for first-order adversarial attacks crafted for the adversarially trained model. This in turn can explain that also the systematically distorted distributions (FGSM) show a rather symmetric behaviour, much like choosing the direction randomly. Then, distortions that stay inside small ϵ-balls around the raw inputs would not compromise the generalization capability of the model [65, 62]. While this behaviour in itself is conclusive and allows to understand the inner workings of the adversarial attacks better, it also endorses the usage of adversarial training to improve the robustness of the model (to first-order adversarial attacks) [64, 63, 111]. With this interpretation, the neural network would not be as vulnerable to mismodellings in simulation, if these can be modelled with noise or adversarial attacks of the kind investigated in this thesis. The visualization of the loss surface [112] could be carried out with a set of training inputs in future studies to back the claim just made about the relation between geometry and model robustness or generalization capability. Surely, more complex attacks would also need more complex defense strategies [63], and it is not clear how sophisticated they need to be in order to model the worst-case scenario due to simulation artefacts, possibly involving correlations. But to summarize the effect of adversarial training, it can be noted that despite the large distortions of several input variables, the model maintains a high performance for jet flavour identification. Possibly the non-limitation of the impact per feature even contributed positively to this property while training with adversarial samples, as the network has learned how to reduce its susceptibility even for unphysical distortions.

4.4 Transferability of adversarial examples

Previous studies have demonstrated that different neural networks are susceptible to the same type of adversarial inputs, even if their setups do not share many similarities besides the classification task for which they are used [6, 63]. For the validation of this property, distorted inputs are transferred between different checkpoints of the training and between the basic and adversarially trained model. The used models in this section do have a common structure, but weights and bias terms do not *a priori* match between them [6, 63].

4.4.1 Adversarial examples crafted from the same epoch, injected to different checkpoints of the model

For this investigation, the FGSM attack utilizes the model after 200 epochs of the training phase, but injects the created adversarial inputs to the three checkpoints at 10, 100 and 200 epochs. The resulting performance on raw and distorted inputs for the discrimination between b+bb and light jets is visualized with the help of ROC curves in Fig. 4.15. All attacks in this example as well as for the comparable case with individually generated adversarial inputs (see Fig. 4.6) use the same type of attack and same limiting parameter (FGSM with $\epsilon = 0.01$). The overall behaviour is comparable to that in Fig. 4.6, where a better performing model is also more susceptible to adversarial attacks. As such, adversarial examples can be transferred between epochs and do still cause a significant loss of performance. But the impact at the epochs 10 and 100, which are not used to create the systematically distorted inputs, is smaller

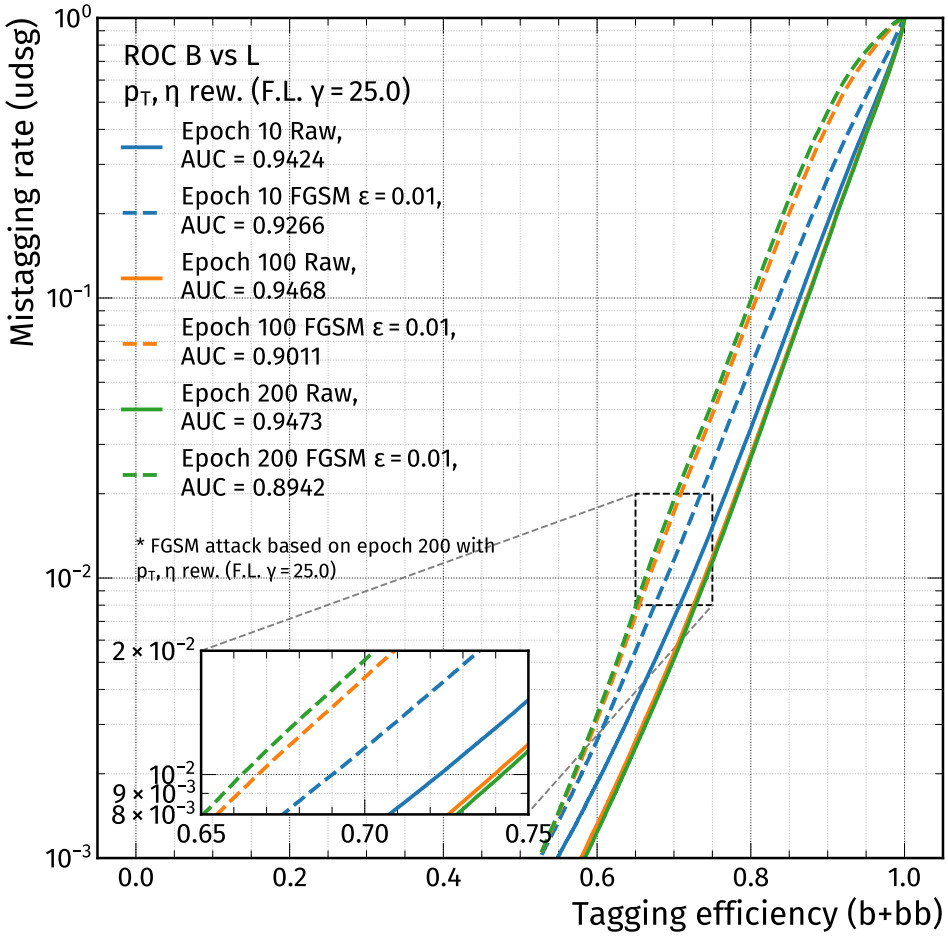

Figure 4.15: ROC curves for the BvsL discriminator, evaluated at epochs 10, 100 and 200 with raw and adversarial inputs, using $\epsilon = 0.01$ for the FGSM attack on the baseline model, crafted from epoch 200 only and injected to all chosen checkpoints. Different checkpoints are displayed in different colours, ROC curves obtained from raw inputs use solid lines, the ROC curves for distorted inputs use dashed lines.

than what can be oberserved in Fig. 4.6, where all adversarial attacks are created individually per state of the model. This can be seen when comparing the AUC values: for the case depicted in Fig. 4.15, the AUC for distorted inputs at epoch 10 is reduced to 0.9266, while an individual creation explicitly for this state of the model reduces the performance to an AUC of only 0.9205. For epoch 100, which is closer to the epoch with which the adversarial inputs are created in this section, the AUC values reach 0.9011 and 0.8993, respectively, showing a smaller difference.

4.4.2 Adversarial examples based on the basic training, injected to the basic and adversarially trained model

Another way to transfer adversarial samples between models utilizes both the basic as well as the adversarially trained model, where the model parameters can potentially differ a lot, related to the different setup of the training loop. This test also uses the BvsL–ROC curves, but the focus is on the difference between the two models, not on different epochs. Therefore the reference against which the following test in Fig. 4.16 can be compared to can be found in Fig. 4.11. In contrast to the previous comparison, the adversarial inputs are now solely based on the basic training at epoch 200 and are injected not only into the basic model, but also the

adversarially trained model. Again, all FGSM attacks use a parameter of $\epsilon = 0.01$ for the distortion. From Fig. 4.16, it is visible that also the adversarially trained model is susceptible to

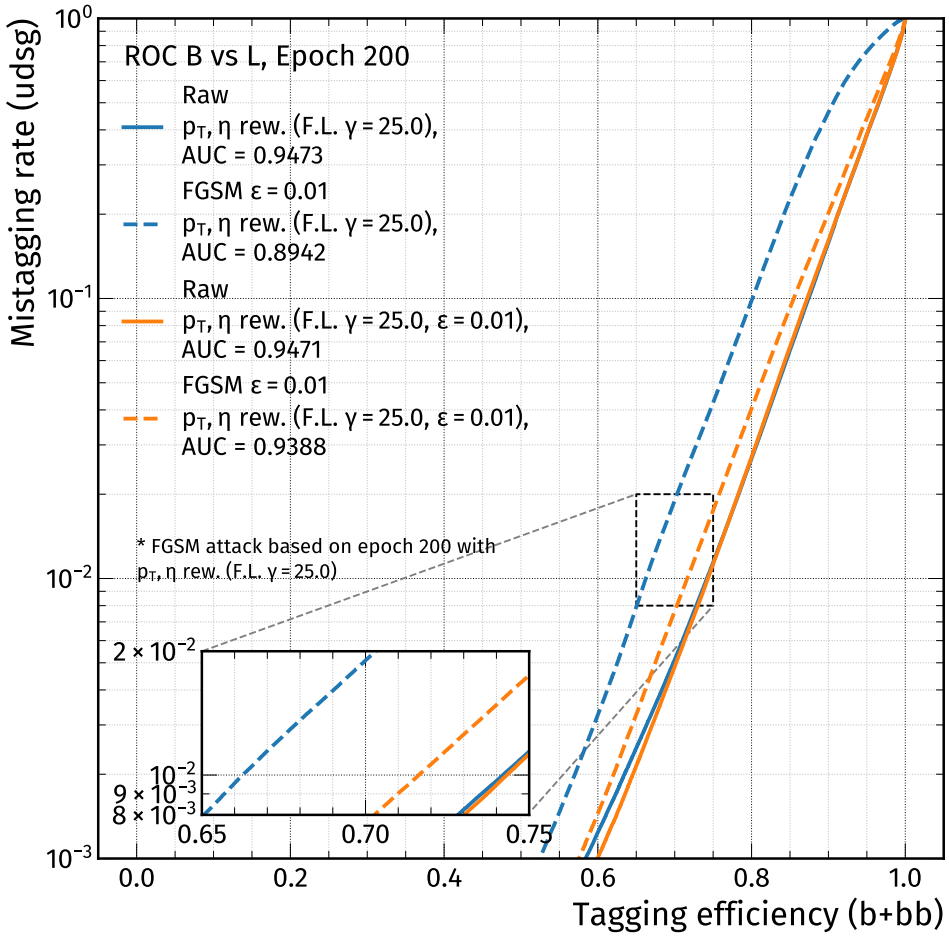

Figure 4.16: ROC curves for the BvsL discriminator, evaluated at epoch 200 with raw and adversarial inputs, using $\epsilon = 0.01$ for the FGSM attack obtained from the baseline classifier, injected to the basic model as well as the adversarially trained model. Raw performance is visualized with solid lines, the performance on distorted inputs uses dashed lines. The basic (adversarial) training is shown in blue (orange), respectively.

the inputs that are crafted from the basic training. Starting with a raw AUC of 0.9471 for the adversarially trained model, the injection of FGSM inputs based on the basic training reduces the AUC to 0.9388. In Fig. 4.11, the usage of an individual FGSM attack for the adversarially trained model results in an AUC of 0.9366, which is slighly less than the performance quoted above. In both cases, the adversarially trained model does show some susceptibility, and the fact that it also negatively reacts to adversarial inputs that are created based on a non-corresponding loss surface suggests that there is a structural similarity between the basic and adversarially trained model [6, 63]. However, also in this case, the individual generation of adversarial inputs has a higher impact on the model performance than inserting external adversarial examples. This test has shown that the claim of a more robust adversarially trained model remains true also when the same inputs (adversarial examples) are inserted for the basic and adversarial training and that it is not only a coincidence related to potential weak adversarial examples for the adversarially trained model.

Currently, a label leaking effect (see Ref. [60]), which refers to a better performance on adversarial examples than on undisturbed data for an adversarially trained model, is not observed. The attack does have an impact, also for the more robust network.

5 Conclusion

5.1 Summary of the results

Concerning the technical aspects related to this thesis, previous investigations conducted in Ref. [9] have now been scaled out to a larger data set, with adjusted cleaning and preprocessing steps. Different training setups have been compared and techniques such as reweighting and the application of the focal loss have been added. The discriminator shapes as well as the key performance metrics mostly resemble those of DeepCSV, although dedicated hyperparameter tuning could improve those measures.

Summarizing the findings up to this point, this thesis has confirmed that adversarial attacks have a concerning impact on model performance, where a subset of the input features shows negligible distortions. On the contrary, with the investigation of rather easy to recognize distortions, the impact of the same attack evaluated on different flavour categories reveals how the discriminating power of physical observables is reduced by adversarial attacks, however, introducing unphysical behaviour for some features. The tradeoff between performance and robustness has been confirmed by investigating the severity of adversarial attacks for different stages of the training, which has highly concerning implications. A defense strategy to improve robustness has been implemented successfully, resisting dedicated first-order adversarial attacks. By comparing the geometric properties of distorted input shapes, the underlying differences for the loss surfaces of the baseline and adversarially trained model can be accessed. Although these modifications can become unphysical, the inversion of the argument is possible as well, namely: despite unphysical distortions for certain variables, the adversarially trained model maintains a high performance for b-tagging. With different parameters and the various setups chosen for the distortions, the susceptibility of deep-learned b-tagging algorithms in their basic form has been showcased, but adversarial training constitutes an encouraging countermeasure, which is expected to be less sensitive to mismodellings in simulation.

Some enhancements that can ameliorate the methods applied during this study are discussed in the following. The outlook section will inspect possible advancements from another, broader perspective, as the first investigations with detector data have been started as well.

5.2 Discussion and possible improvements to the current setup

Restoring the DeepCSV performance while ensuring smooth tagger distributions

Various parameters and choices for the setup influence the model performance and the resulting tagger distributions and discriminators. To ensure comparability to the widely used DeepCSV algorithm, the custom tagger needs a kind of hyperparameter-tuning that includes (non-exhaustive list): the flavour composition, the reweighting setup (focusing parameter, sample weights), the placement of the default values (at the minima or at zero), the optimizer (learning rate decay initilization, momentum), the batch size and the number of epochs, as well as the setup for the adversarial training (parameter ϵ). Also, different network initializations could be probed [2]. The choices of the parameters for this thesis are justified for the reasons discussed throughout the text, especially the introduction of the focal loss, but they

can bring new difficulties and problems that need to be solved in future versions of the training. One key point is that the nominal performance should be comparable to DeepCSV (or get even better) and that the output distributions span the entire range between 0 and 1 with sufficient statistics in every bin, to allow scale factor measurements. Additional studies on the influence of the chosen hyperparameters are given in the appendix (Section A.2).

Applying adversarial attacks and defense strategies of higher complexity while maintaining physical input distributions

In the current setup, distorted inputs are generated by adding Gaussian noise or applying the FGSM attack, with a modification to exclude integer variables or default values. While this is a valid approach to test the susceptibility and improve the robustness when using adversarial samples during the training, one can think of more complex attacks. A first direct addition to the reduced version of the FGSM attack would be to apply a shift of integer distributions as well, making sure to not produce floating point values. The corresponding discussion is already part of Section 4.1.3.

Other techniques to execute adversarial attacks, like implementing an iterative variant of the FGSM attack [64, 60] or the usage of a Generative Adversarial Network (GAN) to obtain adversarial inputs [114, 63] could be a viable next step. In all cases, the distorted input features shall not become unphysical, therefore there needs to be a test that secures compatibility with the raw distributions (related to validation / commissioning [58, 59, 8], see also the constraints mentioned in Section 4.1.3).

As the model investigated during this study already takes high-level features as inputs [8, 4], reducing the input feature space to scale down the vulnerability will most likely result in a much lower performance that is inacceptable for a physics analysis. To some extent, investigating the feature importance might help identify less important variables that could be dropped, which might reduce the susceptibility [63]. But the recent developments in defense strategies offer a lot more room for exploration. Some of them that could be suitable in the case of jet flavour tagging are described below.

Instead of training directly with adversarial examples only, the original loss function can be extended by a penalty term for the malicious data, balanced with a hyperparameter α, as described in Refs. [5] and [63]:

$$\tilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1-\alpha) J(\theta, x + \epsilon \cdot \operatorname{sgn}\left(\nabla_{x_{\text{raw}}} J(y, x_{\text{raw}})\right), y). \tag{5.1}$$

With the additional hyperparameter α, different setups could be tested, although the general paradigm is left unchanged. A similar conclusion can be drawn when trying out other values ϵ for the distortion itself, as so far, only one value has been tested during adversarial training.

Another technique could apply the random noise prior to the FGSM attack, which would also make the defense against such perturbations more difficult [115]. Training against the more sophisticated Projected Gradient Descent (PGD), a multi-step variant of the FGSM attack, can improve the robustness to more powerful adversaries [64]. The improved attack that searches for higher-order adversaries would however also increase the time complexity for the adversarial training, while currently, there is only a constant factor of roughly 1.5 between the basic and adversarial training with the FGSM attack, comparing the time it takes for a single epoch.

The defense strategies are not limited to adversarial training, but in general, methods that produce flatter loss landscapes that are less sensitive to small perturbations are favorable. For example, defensive distillation [116, 63] makes use of two neural networks with the same architecture, where one of them supplies predicted classes for the training of the other network, instead of truth labels. The authors of Ref. [117] mention that although this technique does not

require the generation of adversarial samples, a combination of adversarial training with defensive distillation could complement the defense strategy to be even more effective. A large variety of related regularization techniques could be studied as well, where some directly address the flatness of the loss landscape (e.g. DropAttack [118] and AdvRush [119]).

Subsidiary studies to understand the adversarial attacks better

Some additional tests to get deeper insights into the working principles of adversarial attacks have been started, but this should be continued more thoroughly. Examples for more detailed investigations are given below.

- A feature-importance ranking (e.g. AUC-ranking) could be added to inspect how the discrimination power of each variable changes between the raw inputs and the distorted inputs. So far, this has been started only with a limited number of jets, but could easily be extended.
- The correlations between inputs, before and after the application of an attack, can provide another layer of information on how the attack works or how the discriminating variables are related to each other. The current status on integrating this test is the same as for the AUC-ranking.
- To back the claim regarding the flatness of the loss surface for the adversarial training, contour plots that demonstrate how the loss changes in the vicinity of the raw inputs could be generated. So far, this has just been illustrated with an "educated guess" and would need to be confirmed, possibly by exploring techniques as discussed in Refs. [112, 113, 118, 119].

5.3 Outlook

A major advancement would be to apply the AI safety techniques discussed in this thesis to other models used for jet flavour tagging, like DeepJet. The architecture differs fundamentally, as more low-level inputs are used, where the fully-connected network structure is only one element of the full model, besides convolutional and recurrent layers [51]. With a high-dimensional feature space, the severity of adversarial attacks should become more visible [5, 65], on the other hand, the more complex structure of the network itself could also benefit more from adversarial training [60]. Therefore a comparison with the DeepCSV-like model investigated in this thesis would be interesting.

Ultimately, the model does not need to withstand artificially crafted attacks, but the susceptibility to mismodellings in the simulation needs to be reduced, which should move the derived scale factors closer to one. Consequently, the model needs to be applied not only to simulated jets, but also to data. First steps in that direction have been conducted with the help of a framework originally developed to derive scale factors for c-tagger shape calibration [48]. The progress made so far is documented in Section A.5 of the appendix.

The training itself could be modified such that the performance, robustness and certain requirements on the scale factors enter the optimization in the same framework. Including the obtained scale factors in the training could be achieved by extending the loss function with a penalty term that measures the difference between data and simulation. With paradigms like domain adaptation by backward propagation [120, 121], or, more general, the usage of differentiable programming [122], the training of the network itself could be coupled systematically with the measurement of scale factors. This can introduce inference-awareness [123] by modifying the loss function, while Ref. [109] proposes the inclusion of a test statistic as another input variable (uncertainty-awareness). Both methods, or their combination, could

on the long term reduce uncertainties or bring the scale factors closer to one.

The assumption that adversarial attacks are proxies for simulation mismodellings remains a promising working hypothesis for further studies. A defense strategy like adversarial training can improve the robustness of b-tagging algorithms, which should be considered for future tagger developments.

A Appendix

A.1 Reweighted jet distributions (2D)

When introducing the reweighting techniques, two different target distributions are mentioned: averaging the kinematic quantities p_{T} and η or producing flat distributions of the same (per flavour). The following figures confirm that those target distributions would be matched, if the reweighting was executed by a resampling from the new distributions. Figures A.1 and A.2 contain the twodimensional plots from which the onedimensional visualizations in Section 2.3 are derived.

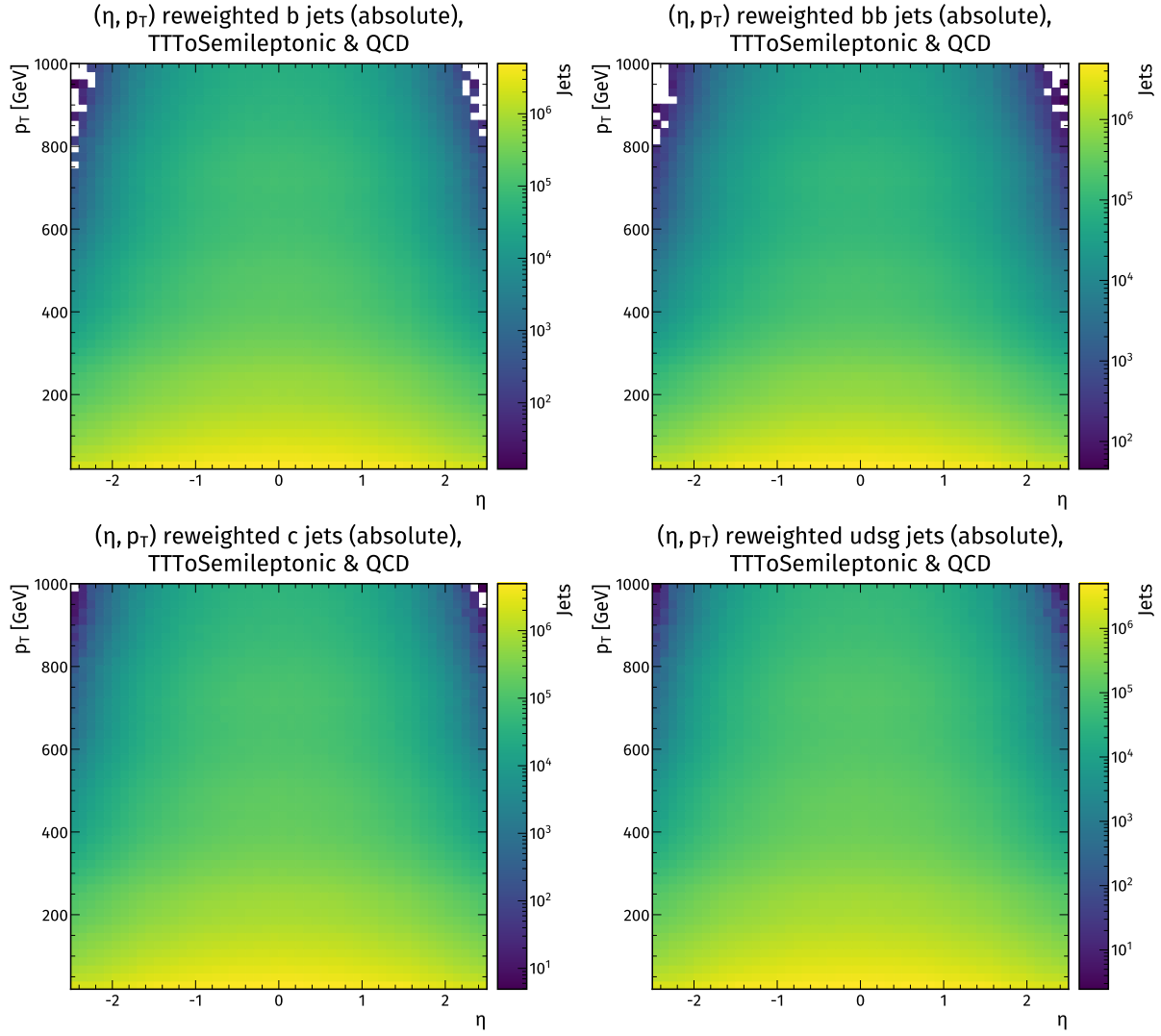

Figure A.1: 2D (η, p_{T}) distributions after reweighting, split by flavour, targeting identical distributions (averaged per flavour).

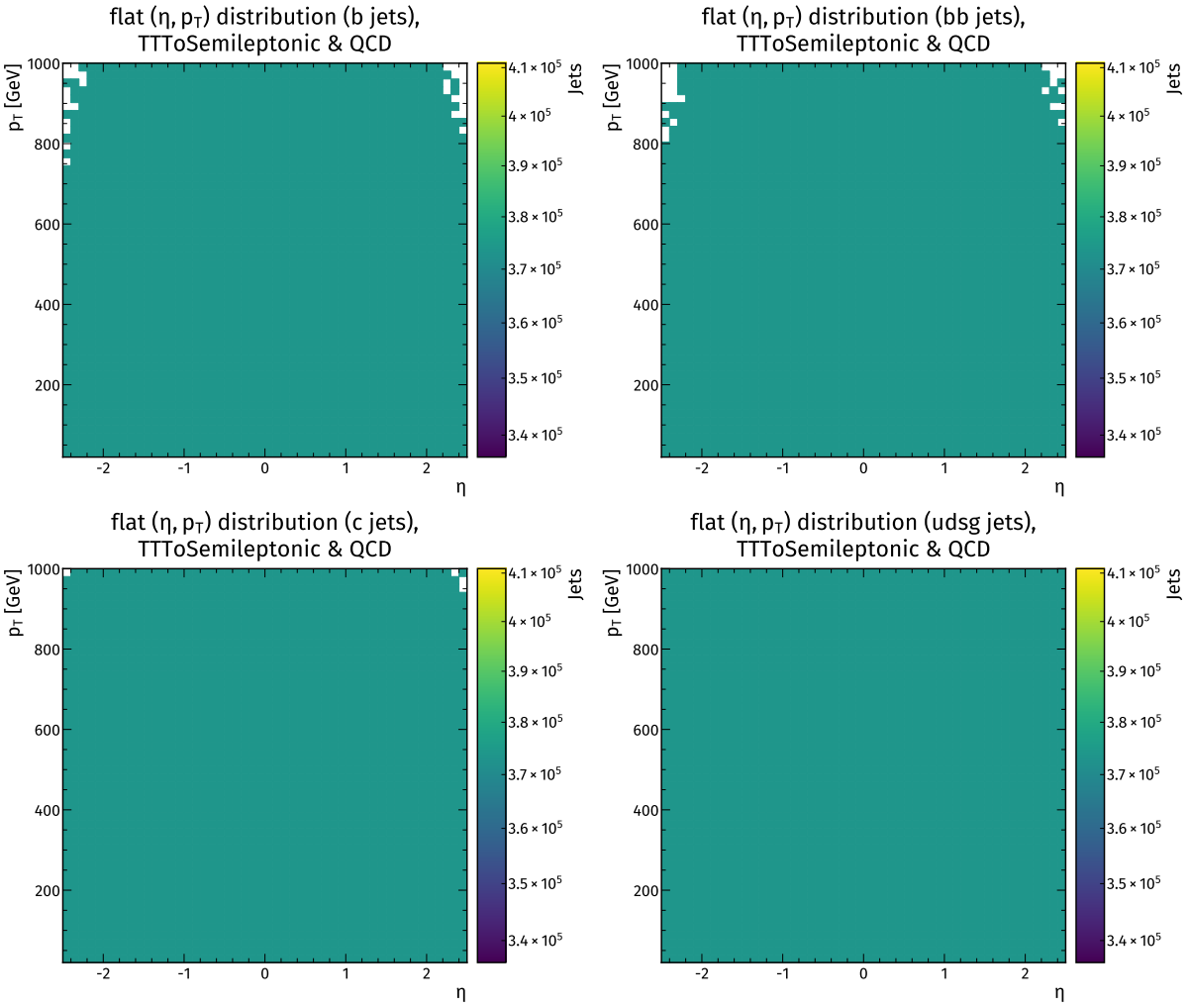

Figure A.2: 2D (η, p_{T}) distributions after reweighting, split by flavour, targeting flat distributions.

A.2 Probing the influence of different hyperparameters or reweighting setups

The hyperparameters chosen for the performance and robustness studies in this thesis are not yet fully optimized. There are various differences related to the discriminator shapes and ROC curves when applying other focusing parameters, for example.

A.2.1 Basic training with $\gamma = 2$

One way to modify how the discriminator shapes are spread out over the entire range between 0 and 1 is to vary the focusing parameter [99], here to the small value of $\gamma = 2$, instead of $\gamma = 25$, as chosen for the main part of this thesis.

Tagger outputs and discriminator shapes

With the smaller focusing parameter, most of the tagger output distributions, as well as the discriminators peak mainly at 0 and 1, leaving only limited statistics in between.

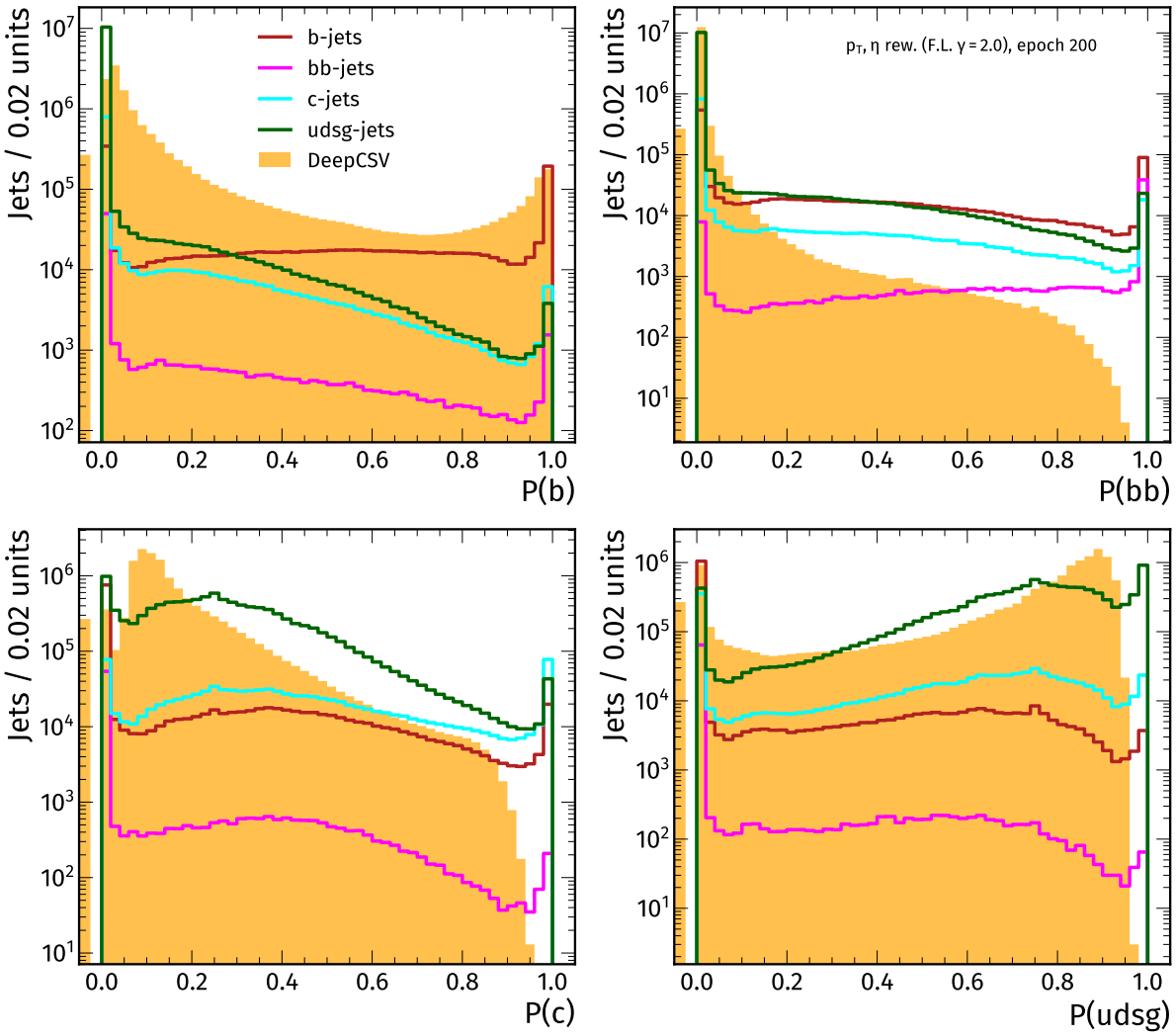

Figure A.3: Tagger outputs, evaluating the basic training with $\gamma = 2$ (split by flavour) and DeepCSV (stacked) on test inputs. Each subfigure represents one output node of the model.

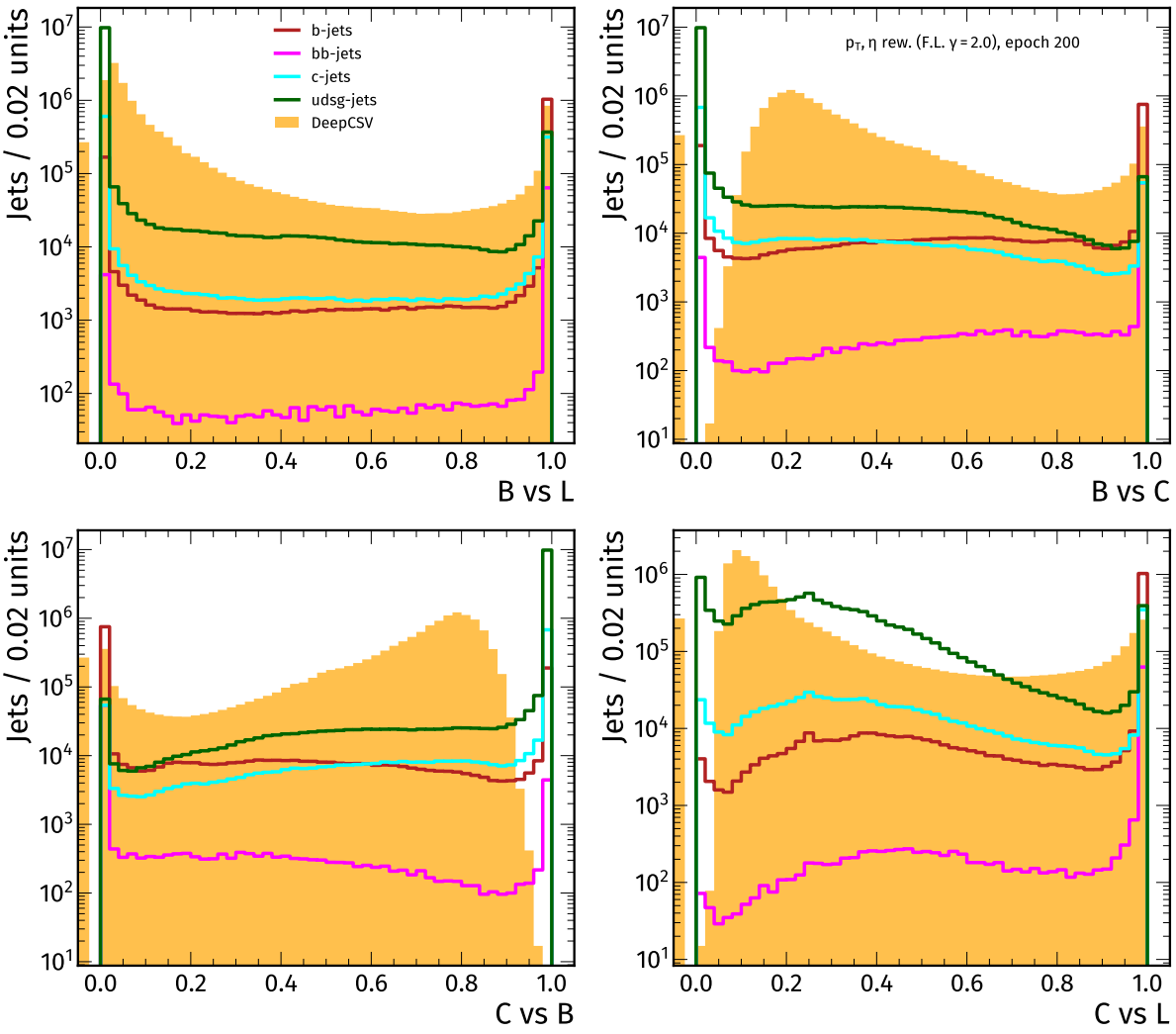

Figure A.4: Discriminators, evaluating the basic training with $\gamma = 2$ (split by flavour) and DeepCSV (stacked) on test inputs. Each subfigure represents one discriminator between two flavours (with category b and bb merged into "B").

ROC curves

Related to the tagger distributions, also the ROC curves show peculiarities when decreasing the focusing parameter. The overall performance for the tagging of b, bb and light jets (against all flavours) is comparable to the case of $\gamma = 25$, but the c-tagging ROC curve (against all flavours) is comparable to DeepCSV now, when using $\gamma = 2$. Again, discriminating one flavour against all others to draw conclusions on the performance is risky (dependent on the flavour composition [8, 48]), therefore, overall the smaller focusing parameter does not improve the performance. Instead, while the AUC might be comparable to the case of $\gamma = 25$, the newly derived ROC curves for $\gamma = 2$ show various cutoffs at relatively high tagging efficiencies, before any attack has been applied to the inputs. For this thesis, the higher focusing parameter provides acceptable ROC curves and shapes without such pathological behaviour. But even there, hyperparameter tuning could provide better occupation of all bins for the discriminators while maintaining a high performance, visualized with ROC curves.

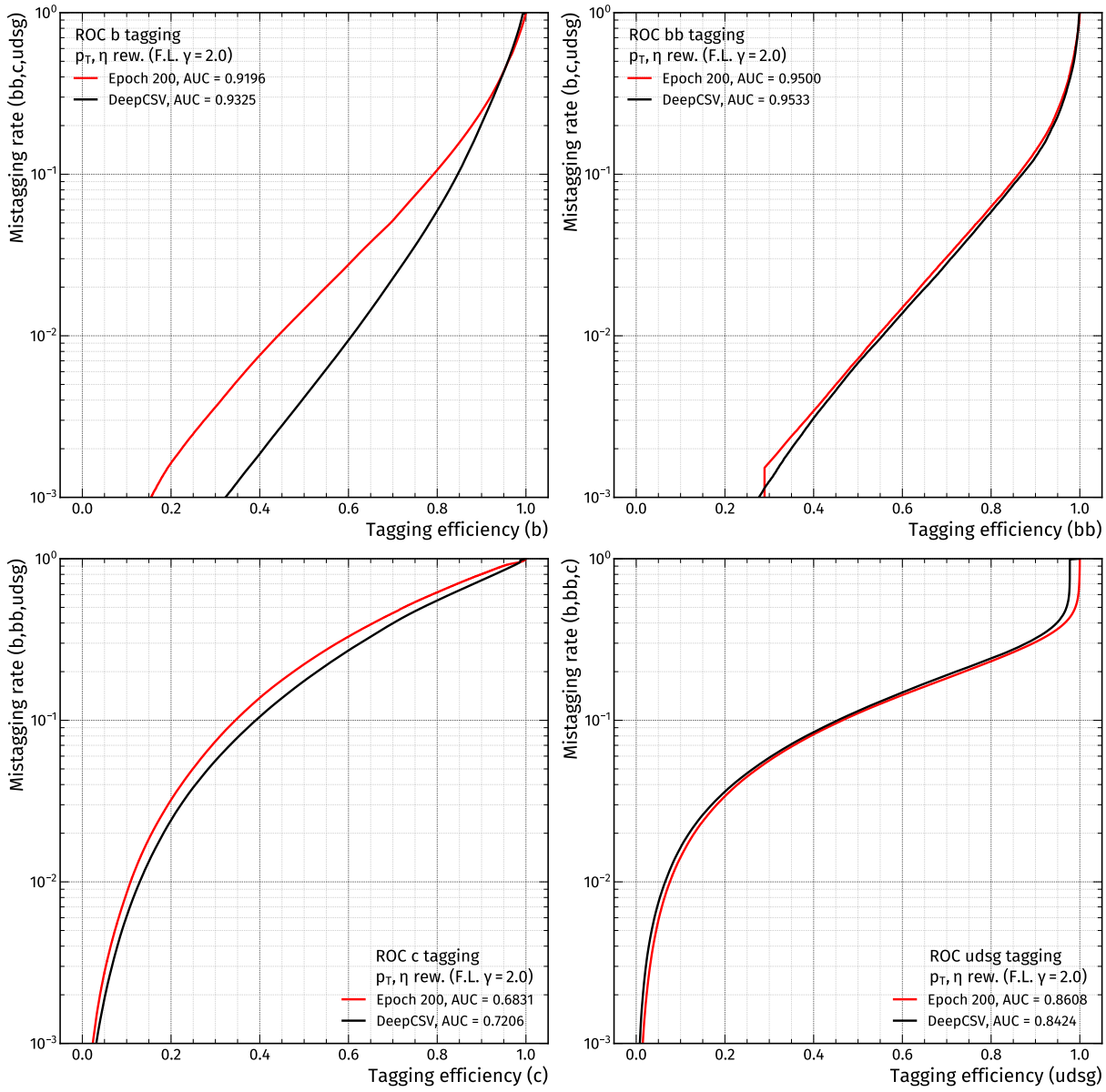

Figure A.5: ROC curves when discriminating each flavour (b, bb, c, udsg) against all other flavours, comparing the custom tagger with $\gamma = 2$ with DeepCSV. The custom tagger is shown in red, the DeepCSV classifier is visualized with black lines.

Remarks on the focusing parameter

Some general remarks on the focal loss [99] follow, by also taking the choice of $\gamma = 25$ for the main part of this thesis into account (see Chapter 3 on nominal performance). The strategy of downweighting easy to classify jets during the training would lead to a good performance for all classes, if the inherent difficulty level for all classes would be distributed identically. Looking at c-tagging (versus *everything*), the large focusing parameter seems to worsen the performance instead of improving it. When taking a look at CvsB or CvsL, it does not look as problematic anymore. A possible cause for the need of further optimization of the reweighting or choosing suitable loss functions is the number of defaults that differs for the different flavours. Many defaulted values for a jet could play too large of a role in defining what it means to be "difficult to classify". The focal loss could pronounce this effect for large values of γ more than desired, introducing another flavour dependency on the performance. When

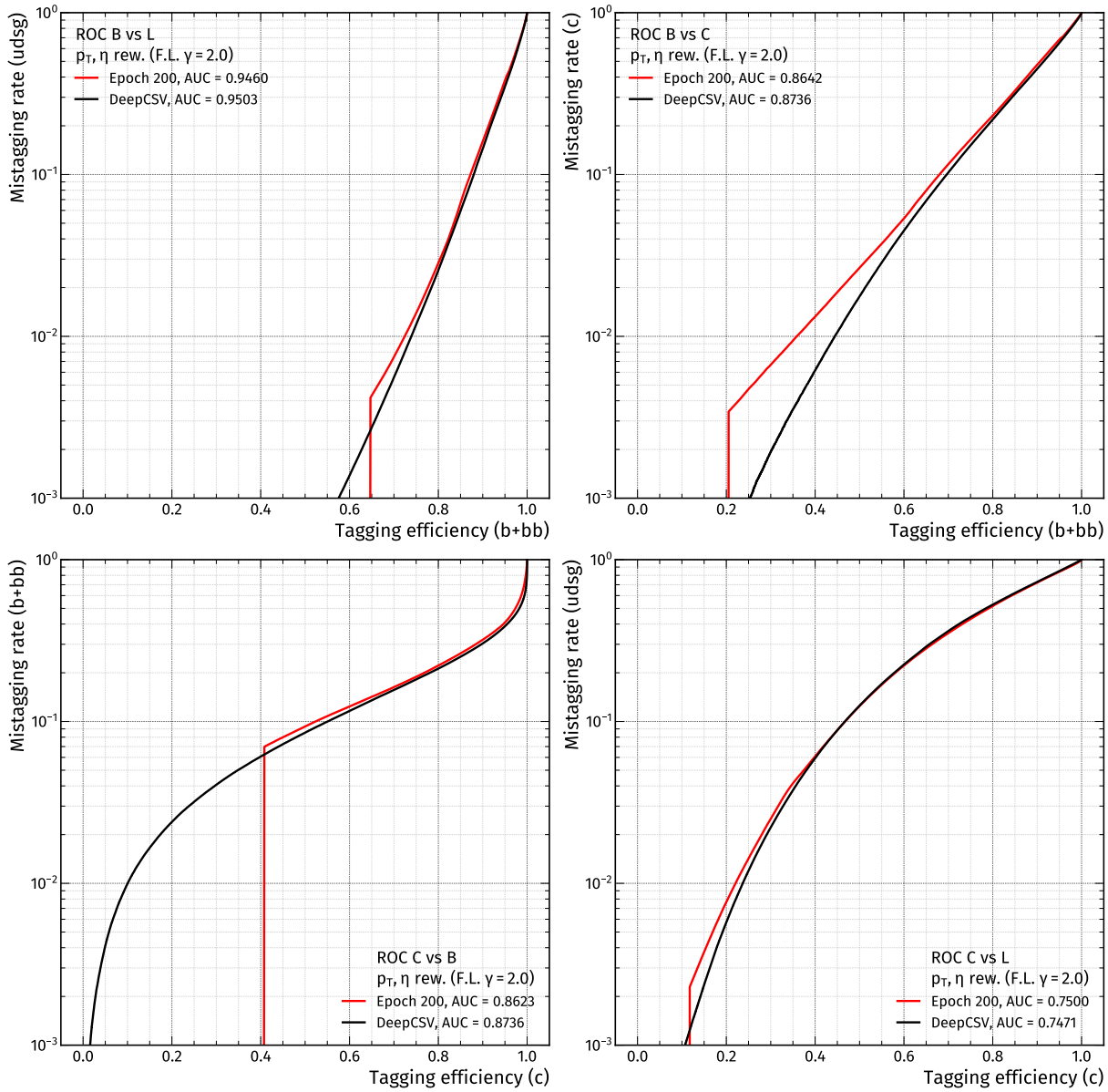

Figure A.6: ROC curves for four different discriminators: BvsL, BvsC, CvsB and CvsL, comparing the custom tagger with $\gamma = 2$ with DeepCSV. The custom tagger is shown in red, the DeepCSV classifier is visualized with black lines.

applying the focal loss, the expectation might be that it helps in improving the performance for c jets (which is *a priori* more difficult due to c jets having "intermediate" properties when compared to b or light jets). But the effects mentioned above could overcorrect the jets with many defaults in the majority class, ignoring the inherent difficulty of c jets. Finding suitable corrections to the sample or class weights based on how many input variables of the jet are set to a default value seems desirable, especially when using large focusing parameters. It is evident that the focal loss with large focusing parameters does not produce the problematic shapes that are basically only filled at the edges at 0 and 1, but now some distributions show cutoffs. This is inconvenient for the derivation of scale factors, which needs statistics in the entire range.

A.2.2 Basic training with flat reweighting in p_{T} and η

Another way to modify the training setup is to apply the flat reweighting for the transverse momentum and pseudorapidity of the jet (instead of averaging) [8, 94]. The tagger distributions are very similar to the ones shown before ("average-between-flavours"-reweighting), and to demonstrate that the performance with flattened kinematic quantities is marginally worse, ROC curves for the BvsL discrimination are given in Fig. A.7.

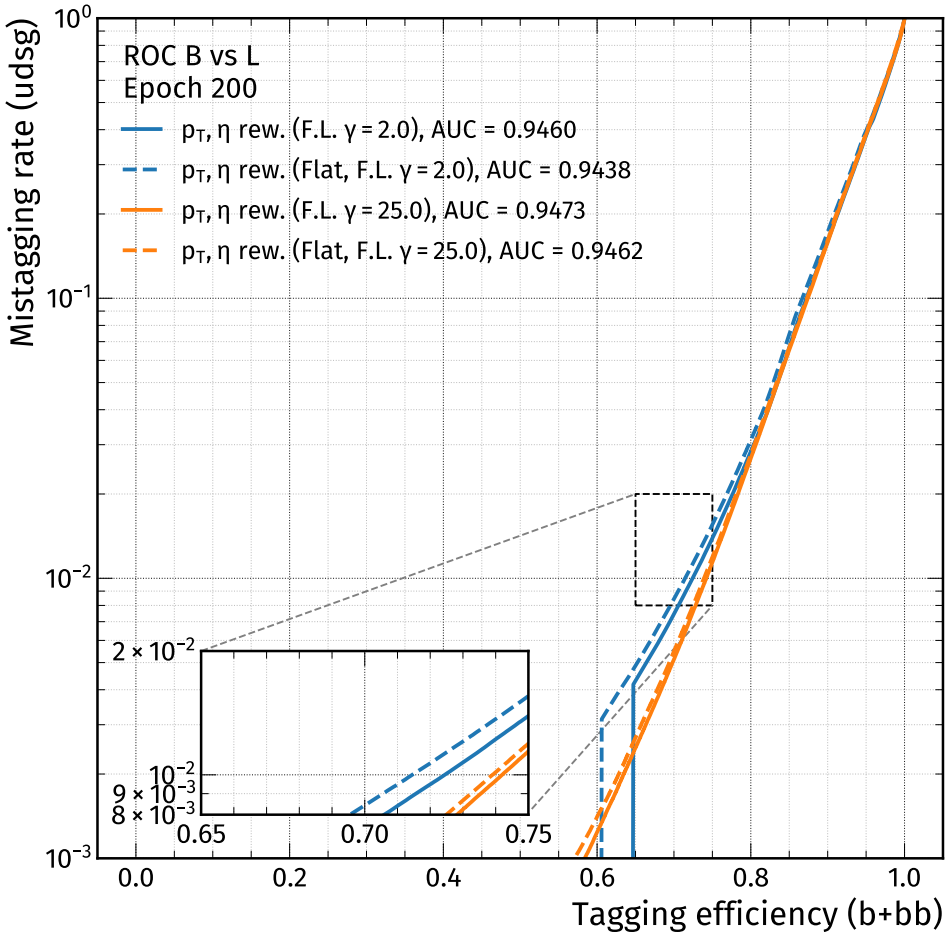

Figure A.7: ROC curves for the BvsL discriminator, evaluated for different reweighting techniques. The blue curves represent the training with a focusing parameter of $\gamma = 2$, while the orange curves correspond to $\gamma = 25$. Solid lines show the performance for the basic reweighting method that averages the kinematic quantities (pseudorapidity and transverse momentum of the jet) between different flavours, while the dashed lines correspond to flat p_{T} and η distributions after reweighting.

A.3 Additional input shapes with adversarial samples (FGSM attack)

The following figures show additional input features (all remaining jet and secondary vertex variables, as well as the first track variables (second to sixth track have a qualitatively similar behaviour)), although only four of them will be discussed in more detail. When taking a look at the histograms displayed in Fig. A.8, some distinct features become visible. Starting with the signed impact parameter significance (3D, first track above charm threshold), it can be seen that the default value is not perfectly chosen. This is due to the computation of the minima that utilizes the average minimum value of several files [92]. It is therefore not necessarily a global minimum for all data sets, but the requirement of placing the default outside the main (central) part of the distribution is still fulfilled. The above mentioned input variable is visualized also in the main part of this thesis (see Fig. 4.13), but for a small selected range where the distortions are easily visible. Compared to the investigation carried out there

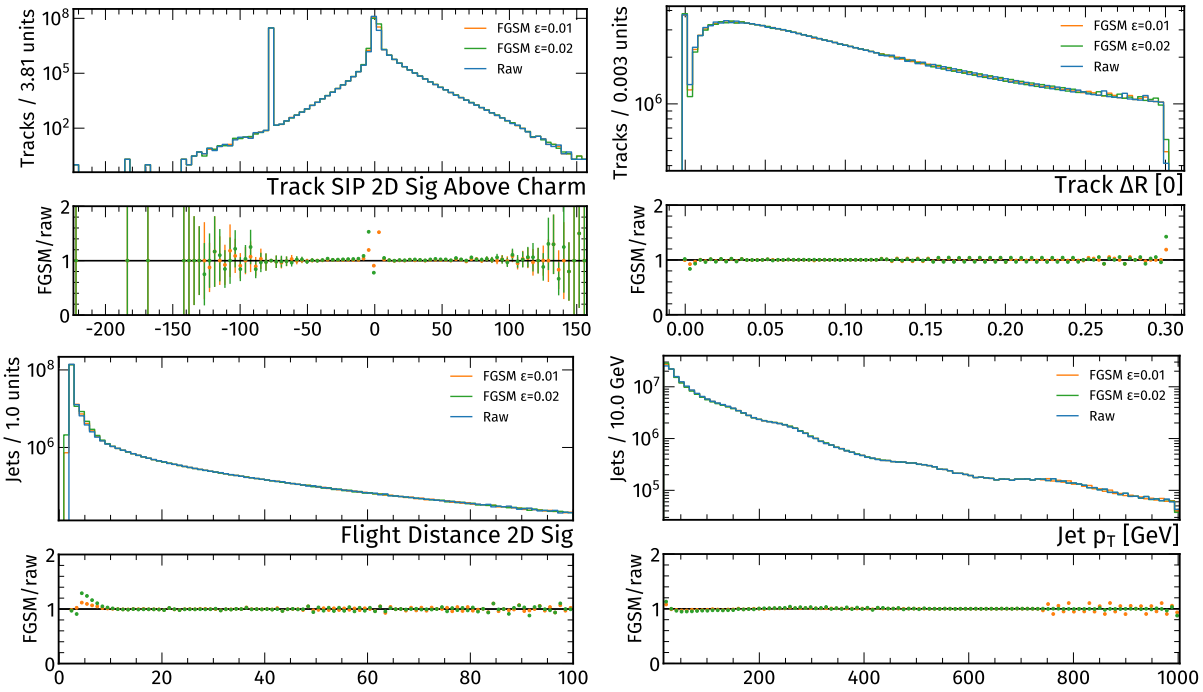

Figure A.8: Systematically distorted inputs that show distinct patterns also for large x axis ranges. For the adversarial attacks, the parameters are either moderate ($\epsilon = 0.01$) or rather large ($\epsilon = 0.02$).

(split by flavour), also for the inclusive case over all flavours the distorted inputs show a disagreement with the raw distribution. This tension is not as pronounced as for the enlarged ("zoomed-in") region, which emphasizes the importance of defining a suitable range beforehand. The other three observables shown in the figure above all reveal a certain oscillation pattern that starts to appear at a given value. With the subfigure that shows the angular distance ΔR between the first track and the jet, this property can be explained from the intervals between which this pattern occurs, namely intervals between consecutive powers of 2. For example, the range between $2^{-3} = 0.125$ and $2^{-2} = 0.25$ does show an oscillation pattern for the attack with $\epsilon = 0.02$ that alternates between consecutive bins, while for the range above $2^{-2} = 0.25$ the pattern has twice that oscillation length. This is related to the floating point precision, which adapts increasing intervals between the storable numbers in the binary system [124, 105]. Each time the binary representation switches to another, higher power of 2,

also this oscillation pattern changes frequency. The adversarial attack is a translation of the raw distribution, therefore this pattern is transported consequently (for example to the positive side, like it is the case for subfigure in the top right of Fig. A.8). It can be questioned whether the `Float16`-precision is sufficient for all variables and whether the binning is too fine [125]. For this, the uncertainties and necessary scale for all observables need to be known. As another example of a variable with high discrimination power [8], the flight distance sig-

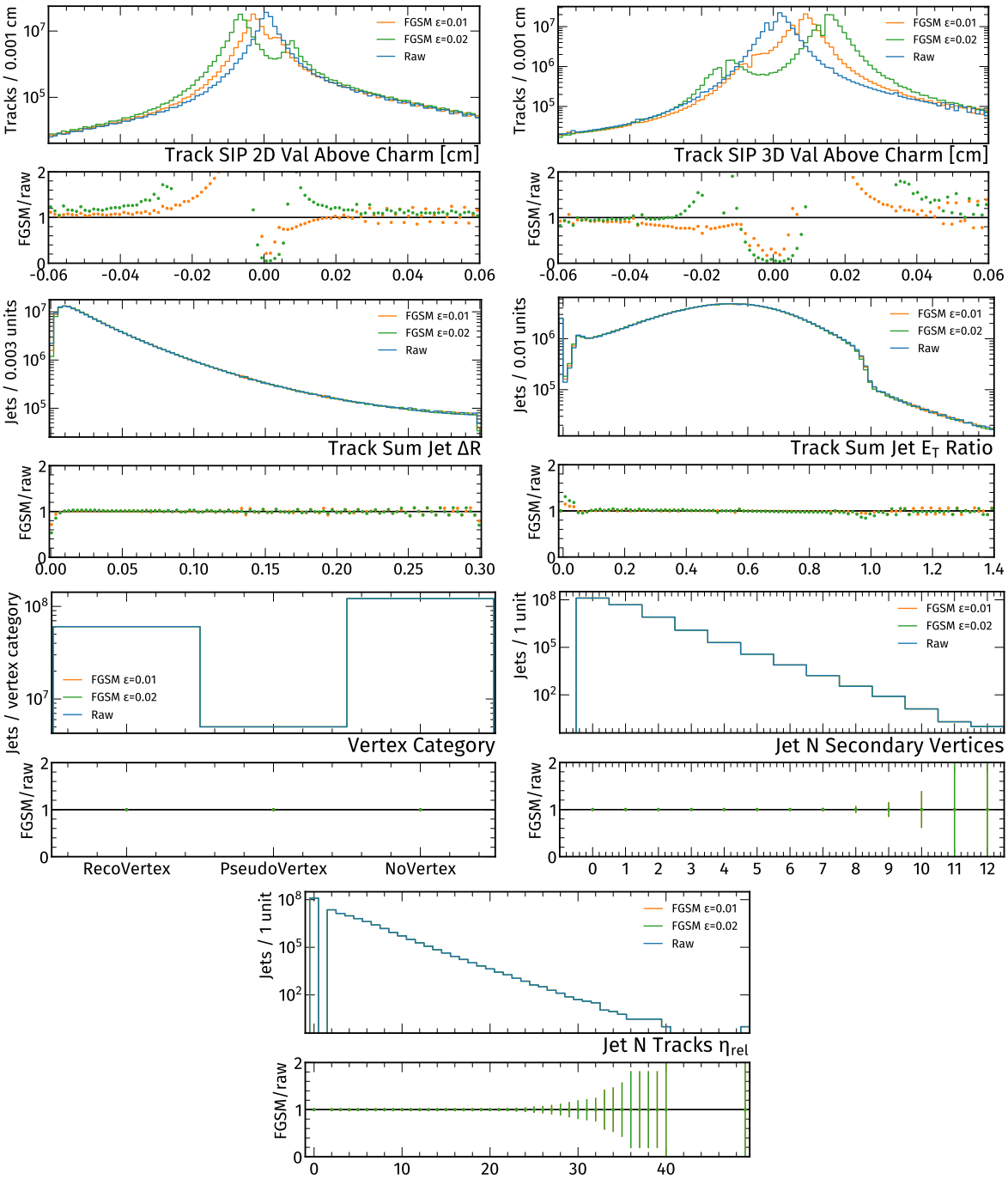

Figure A.9: Remaining jet variables with systematically distorted inputs. For the adversarial attacks, the parameters are either moderate ($\epsilon = 0.01$) or rather large ($\epsilon = 0.02$). Note how the categorical or integer features are not modified at all, by construction.

nificance (2D) of the secondary vertex is given in the lower left. Besides the oscillation that occurs in the low statistics region, there is also some systematic shift at significances between

around 5 and 10 towards larger values. As demonstrated previously, the attack does shift the raw inputs mainly into originally less abundant regions (depending on the flavour). Considering that most jets in the data set are light jets, this would exactly match this behaviour. Normally, a large flight distance corresponds to heavy-flavour jets because the heavy hadrons decay later. This is therefore an example of a variable that shows a systematic impact of the adversarial attack, without the need to split by flavour and in a rather large range. For the transverse momentum of the jet, no such large systematic shift is expected. With the help of the reweighting step, not much discriminating power is expected from the purely kinematic quantities [8, 94]. The distribution shows an oscillatoric pattern at high p_{T} values, but only for the smaller of the two distortions ($\epsilon = 0.01$), it looks like this magnitude of the impact is "in resonance" with the inherent pattern of the raw distribution due to the floating point precision. A slight trend is visible between $0\,\mathrm{GeV}$ and $250\,\mathrm{GeV}$, there is a tiny positive slope for the ratios. This could be a hint that the reweighting to identical, but non-flat shapes per flavour is not enough.

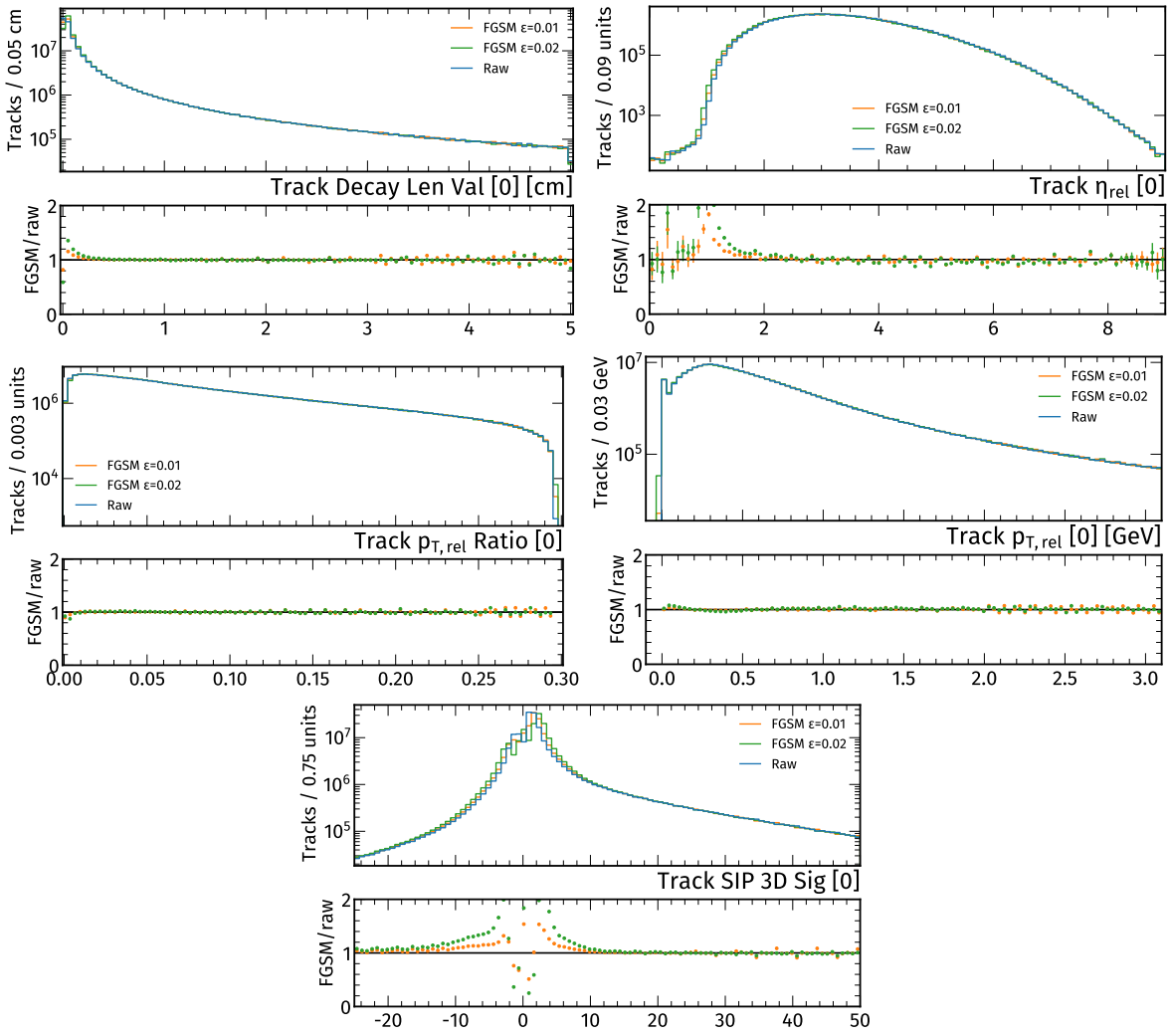

Figure A.10: Remaining first track variables with systematically distorted inputs. For the adversarial attacks, the parameters are either moderate ($\epsilon = 0.01$) or rather large ($\epsilon = 0.02$). The qualitative behaviour for the second to sixth track is similar and is not displayed here.

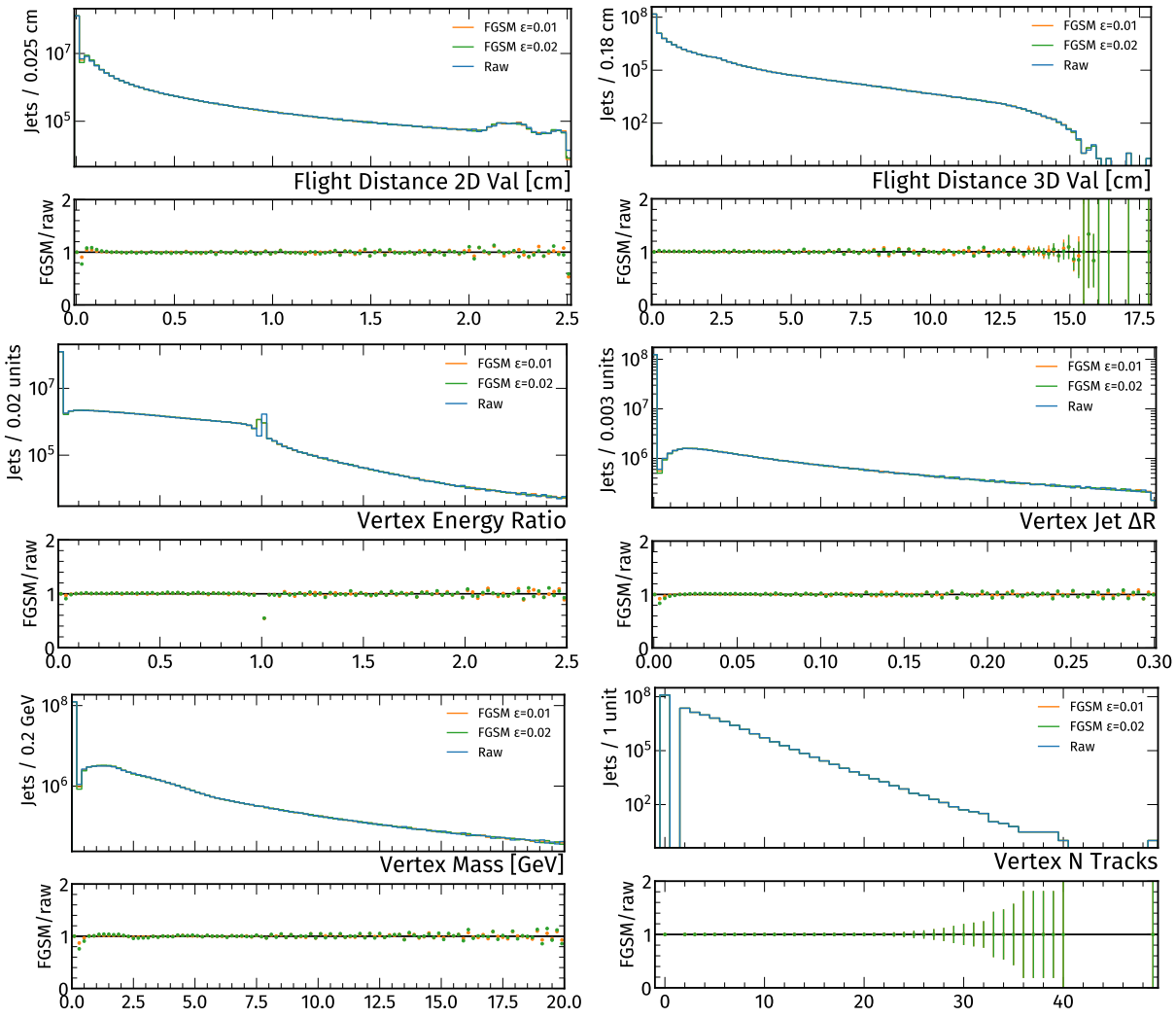

Figure A.11: Remaining secondary vertex variables with systematically distorted inputs. For the adversarial attacks, the parameters are either moderate ($\epsilon = 0.01$) or rather large ($\epsilon = 0.02$). Note how the integer variable that counts the number of selected tracks associated with the secondary vertex is not modified at all, by construction.

A.4 Additional comparisons of performance for the adversarially trained model

A.4.1 Raw performance (adversarial training), compared to DeepCSV

Tagger outputs and discriminators

The following comparisons show how the tagger distributions (and discriminators) look for the adversarial training, using raw inputs. Overall, the shapes resemble those obtained from the basic training, but the discriminators for the adversarially trained model are spread slightly more between the possible values between 0 and 1.

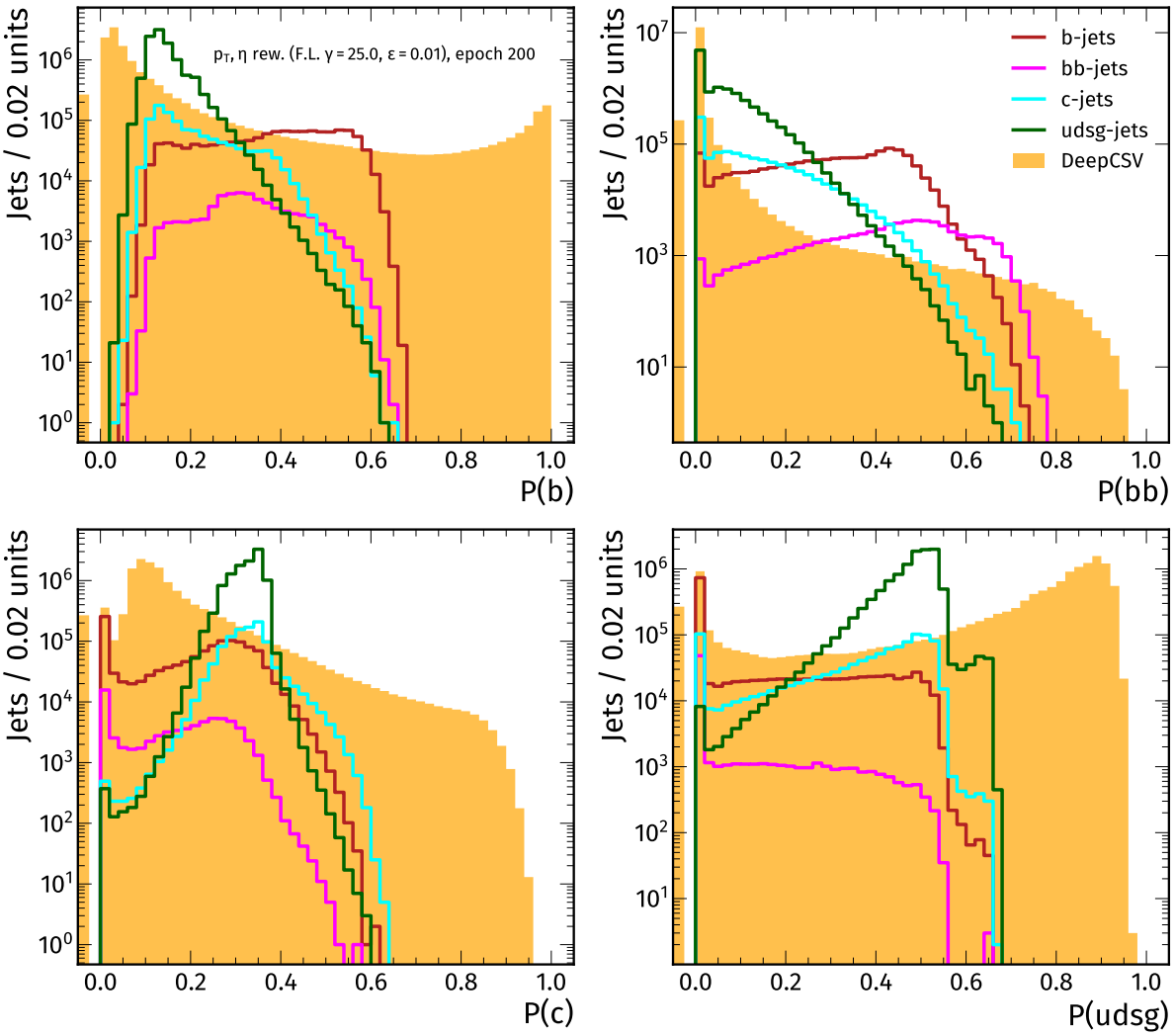

Figure A.12: Tagger outputs, evaluating the adversarial training with $\gamma = 25$ (split by flavour) and DeepCSV (stacked) on test inputs. Each subfigure represents one output node of the model.

ROC curves

Also the ROC curves are very similar to the basic training, both for the "X versus All" and the "X versus Y" case. The reweighting strategy and focusing parameters are identical, therefore

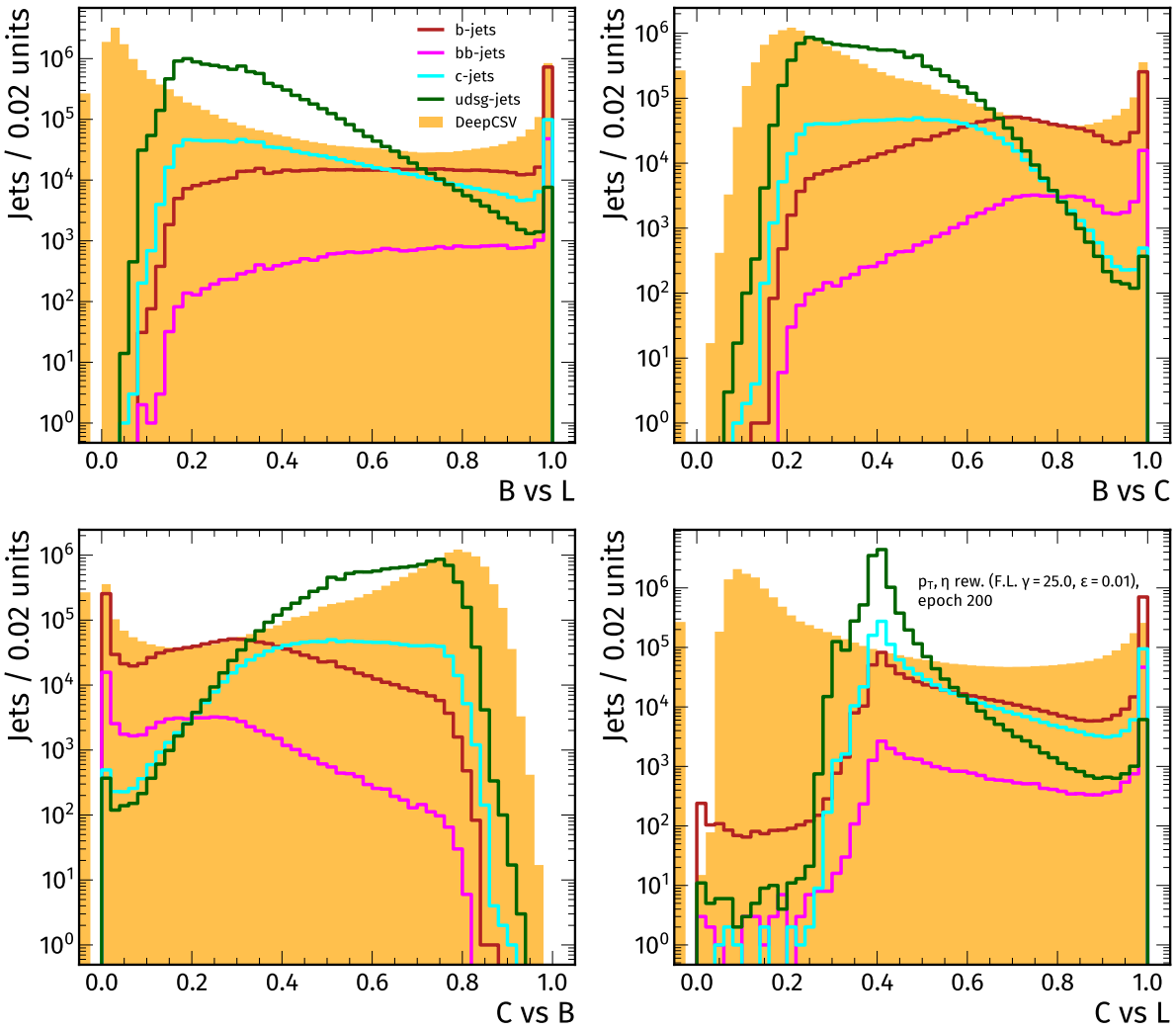

Figure A.13: Discriminators, evaluating the adversarial training with $\gamma = 25$ (split by flavour) and DeepCSV (stacked) on test inputs. Each subfigure represents one discriminator between two flavours (with category b and bb merged into ″B″).

it looks like the adversarial training has a much smaller impact on the raw performance than a varying hyperparameter setup would have (see Section A.2).

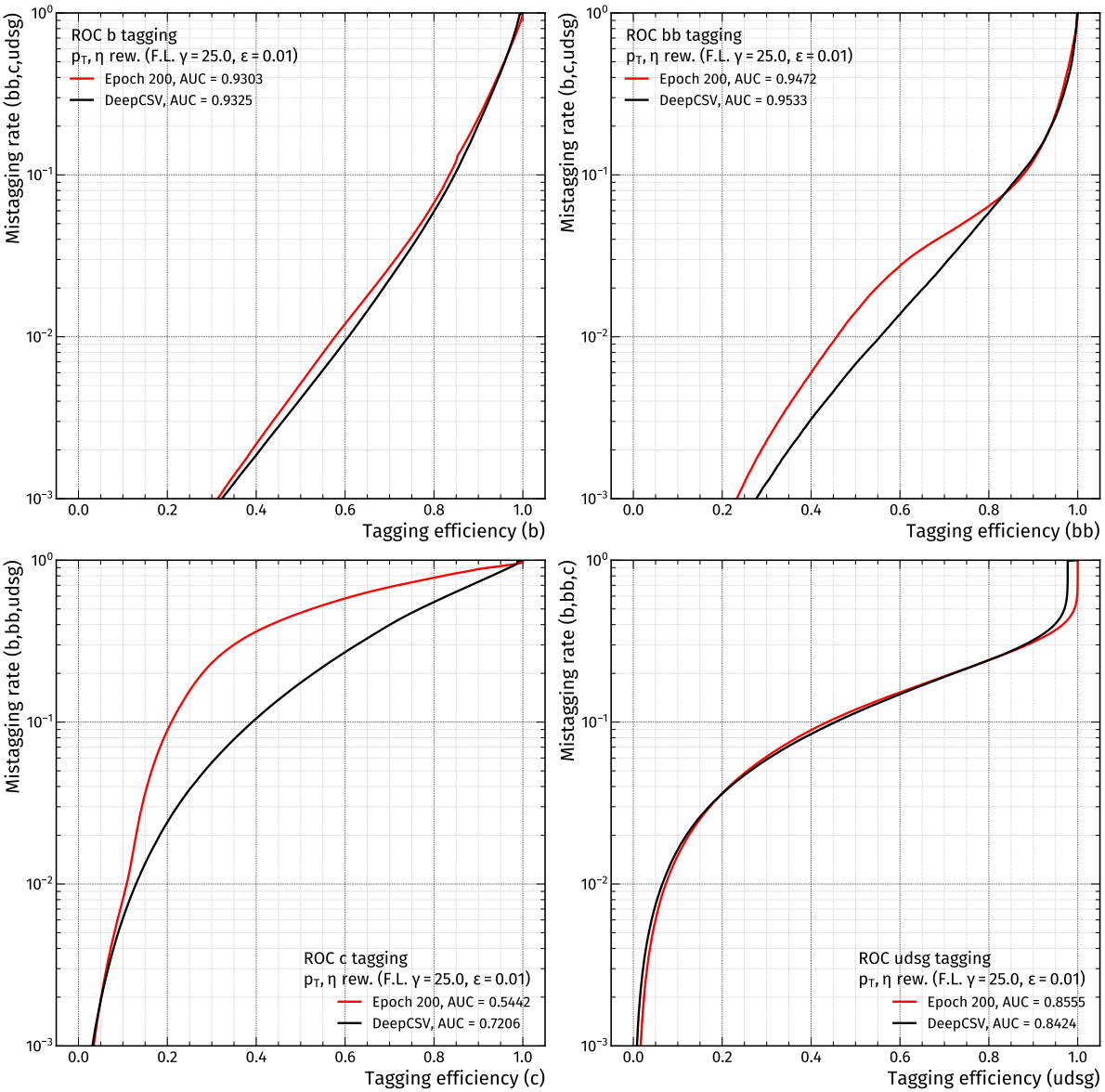

Figure A.14: ROC curves for the adversarial training with $\gamma = 25$, when discriminating each flavour (b, bb, c, udsg) against all other flavours. This metric depends on the flavour distribution present in the test sample. Therefore, it is not safe to measure the performance with this method. The custom tagger is shown in red, the DeepCSV classifier is visualized with black lines.

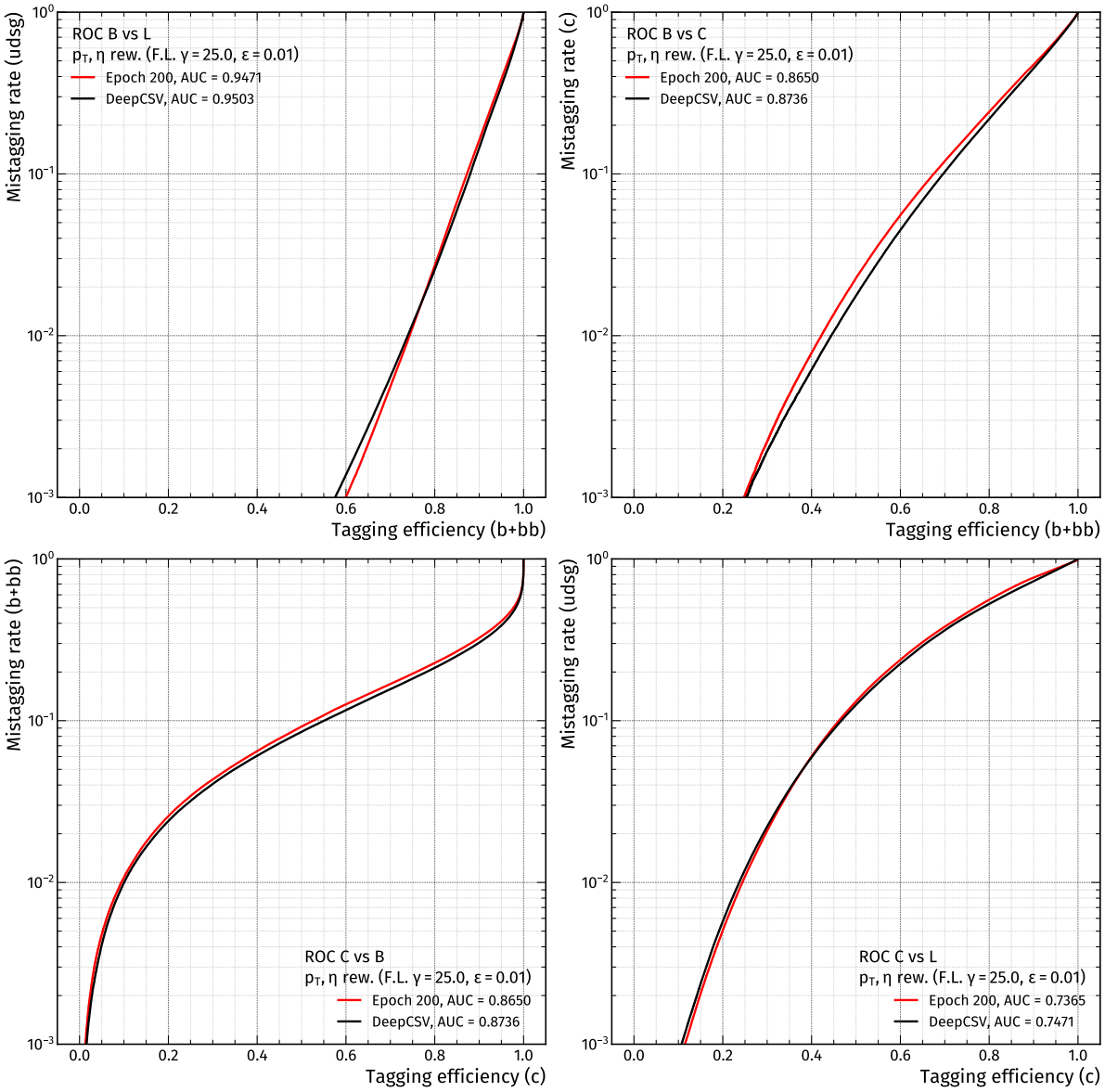

Figure A.15: ROC curves for the adversarial training with $\gamma = 25$, for four different discriminators: BvsL, BvsC, CvsB and CvsL. The custom tagger is shown in red, the DeepCSV classifier is visualized with black lines.

A.4.2 ROC curves (Gaussian noise)

When manipulating the inputs with Gaussian noise, the impact on the performance for the basic training is already small (see for example Fig. 4.7), for the adversarially trained model the impact is even less. Distortions as large as $\sigma = 0.1$ as shown on the left side of Fig. A.16 reduce the BvsL–AUC by less than $1\,\%$, where the distorted input shapes would already show unphysical features, as explained earlier. The general behaviour (larger distortion with Gaussian noise $\rightarrow$ worse performance) is also observed for the adversarially trained model. On

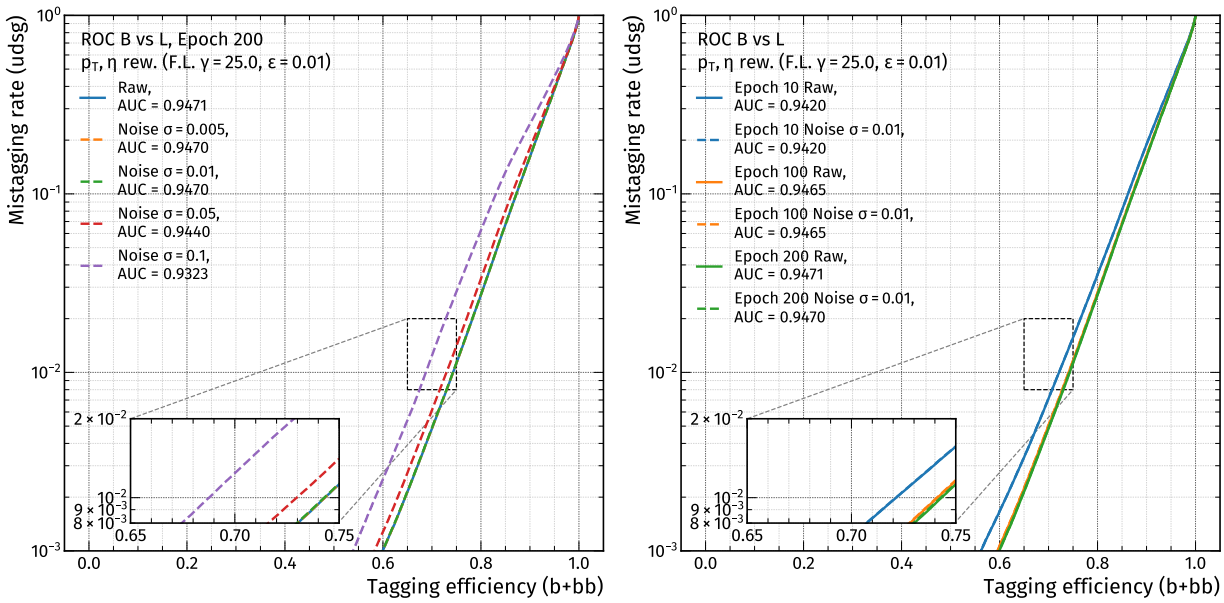

Figure A.16: ROC curves for the BvsL discriminator, evaluating the adversarially trained model with raw and randomly distorted inputs (Gaussian noise), either with a fixed epoch with different parameters σ, or with a fixed parameter σ, but evaluated at different checkpoints of the training. The left subfigure uses epoch 200, different colours represent different magnitudes of the distortion. On the right, a fixed parameter $\sigma = 0.01$ (dashed lines) is utilized in comparison with raw inputs (solid lines), for different epochs in different colours.

the right subfigure, the varying impact with the same magnitude of random noise is investigated for different epochs (with the same setup as for all previous checks of that kind), now for the adversarial training. A dependence on the number of epochs can not be visualized for this type and magnitude of the distortion here, the usage of larger parameters (unphysical) would be necessary to even see the impact for the presumably most vulnerable epoch 200. With reasonable parameters for Gaussian noise, Fig. A.17 shows how small the impact is for both the basic and the adversarial training. Though from the denoted numbers, the AUC drops by around 0.001 for the basic training and by roughly 0.0001 for the adversarial training, the differences are so small that compared to unrelated modifications to the setup like using only a specific part of the phase space or choosing a slightly different test sample would likely outnumber these differences, and again, systematic uncertainties would cover such slight changed entirely [8, 48].

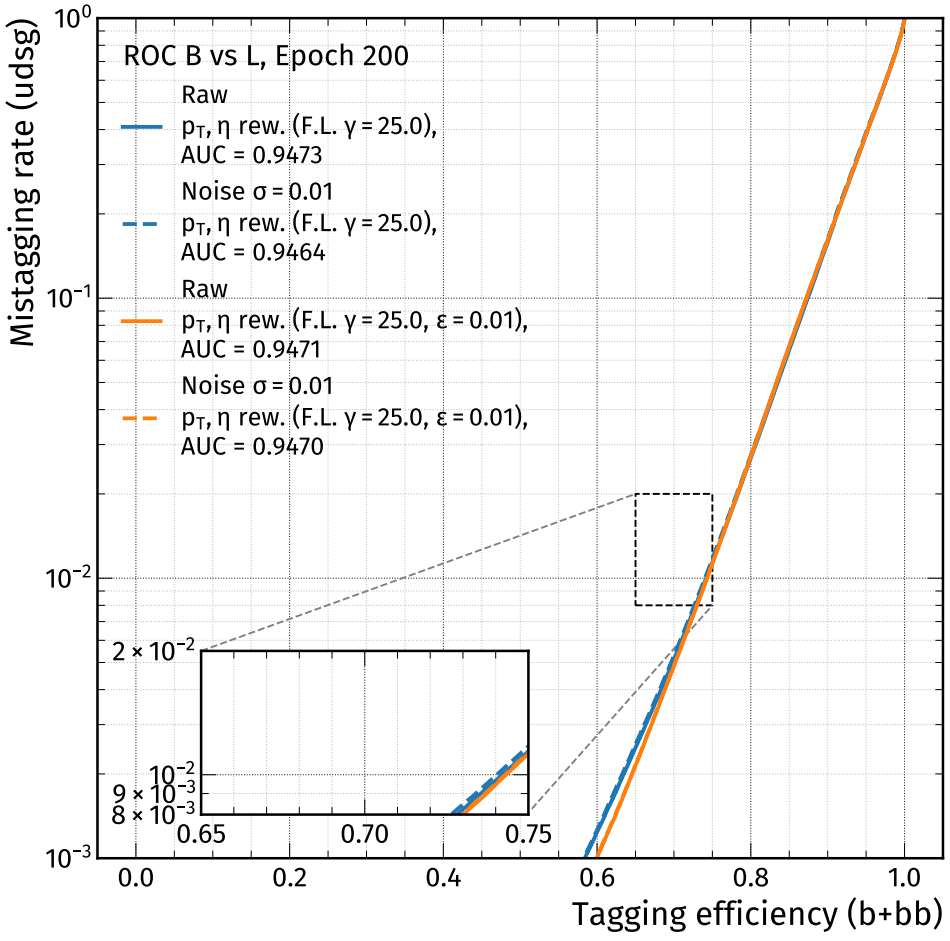

Figure A.17: ROC curves for the BvsL discriminator, evaluated at epoch 200 with raw and distorted inputs, using $\sigma = 0.01$ for the Gaussian noise term, inserted to the basic model as well as the adversarially trained model. Raw performance is visualized with solid lines, the performance on distorted inputs uses dashed lines. The basic (adversarial) training is shown in blue (orange), respectively.

A.4.3 AUC ratios (Gaussian noise and FGSM)

Taking the ratios between distorted and raw performance for the adversarial training, measured with the BvsL–AUC over epoch, it can be noticed that the impact stays approximately constant over the full training phase (see Fig. A.18). Fluctuations between subsequent epochs are more evident than a general trend for the dependence on epoch. The absolute scales differ between noise and FGSM, as has been observed for the basic training already.

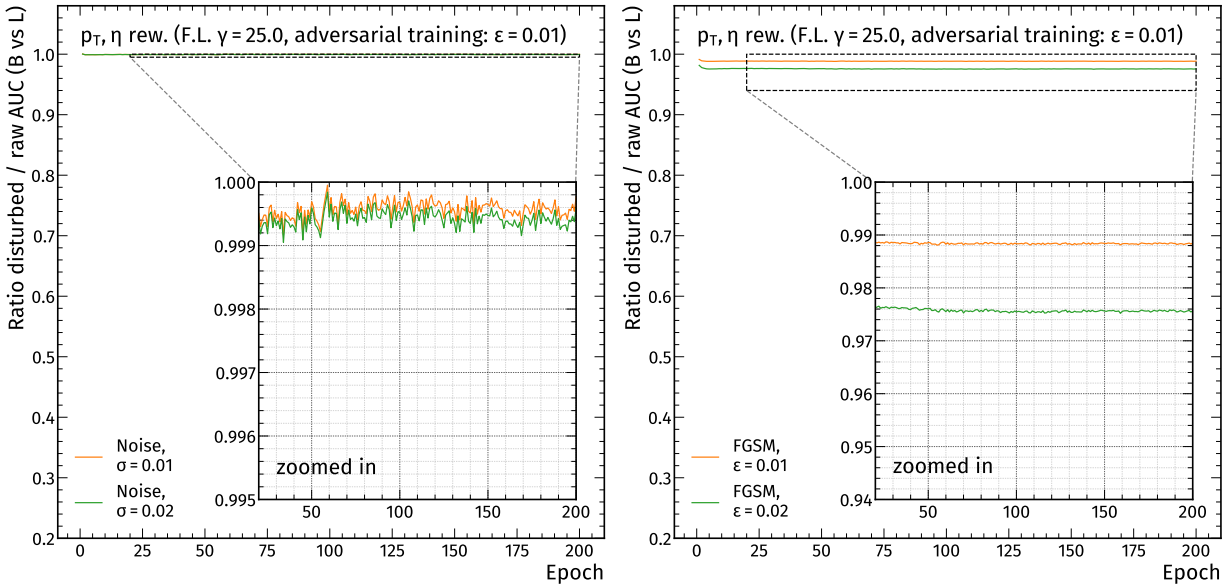

Figure A.18: Evolution of the ratio between distorted and raw BvsL–AUC with the number of epochs for the adversarially trained model, evaluated on raw and distorted inputs (adding noise with $\sigma = 0.01$ and $\sigma = 0.02$ (left) or by using the FGSM attack with $\epsilon = 0.01$ and $\epsilon = 0.02$ (right)).

A.5 Investigating the basic and adversarial training with the help of a scale factor framework

The trained models are evaluated on detector data, recorded by CMS in 2017, as well as on simulated events containing jets (similar to Ref. [48]). Three selections are taken into account, each enriching one particular flavour (b, c and light), but considering only the muon channel. The setup for the scale factor derivation uses a dedicated analyzer script for the different selections, a tool to obtain stacked histograms, as well as the implementation of an adaptive fit algorithm, originally created for Ref. [48, 126], where the focus is on c tagger shape calibration. For this thesis, most configurations replicate the ones used for the DeepCSV tagger as studied in Ref. [48], the difference being a slightly reduced collection of samples, as listed in Table A.1, and the usage of files obtained from the `PFNano` framework [86].

Table A.1: Used samples to enrich the three flavours b, c and light with simulated events and with data recorded by CMS in 2017 ($\sqrt{s} = 13\,\text{TeV}$, $\mathcal{L}_{\text{integrated}} = 41.54\,\text{fb}^{-1}$). More information on the samples and selections can be found in Refs. [48, 126].

Samples for the W+c & semileptonic $t\bar{t}$ selection (c or b jet enriched)	
MC	`DYJetsToLL_M-50_TuneCP5_13TeV-madgraphMLM-pythia8`
	`TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8`
	`TTToHadronic_TuneCP5_13TeV-powheg-pythia8`
	`TTTo2L2Nu_TuneCP5_13TeV-powheg-pythia8`
	`WJetsToLNu_TuneCP5_13TeV-madgraphMLM-pythia8`
	`ST_s-channel_4f_leptonDecays_TuneCP5_13TeV-amcatnlo-pythia8`
	`ST_t-channel_top_4f_inclusiveDecays_TuneCP5_13TeV-powhegV2-madspin-pythia8`
	`ST_t-channel_antitop_4f_inclusiveDecays_TuneCP5_13TeV-powhegV2-madspin-pythia8`
	`ST_tW_top_5f_inclusiveDecays_TuneCP5_13TeV-powheg-pythia8`
	`ST_tW_antitop_5f_inclusiveDecays_TuneCP5_13TeV-powheg-pythia8`
Data	`SingleMuon`
Samples for the Drell–Yan (DY)+jets selection (light jet enriched)	
MC	`DYJetsToLL_M-50_TuneCP5_13TeV-madgraphMLM-pythia8`
Data	`DoubleMuon`

To adapt this framework for the custom taggers under investigation, mainly the analyzer part is extended such that raw tagger scores as well as discriminators can be compared later, using either the basic or adversarial training, evaluated at different checkpoints and utilized either with raw inputs or with an applied distortion (noise or FGSM). Therefore the scripts that run the analyzer are supplemented with an inference step with the custom model, which takes raw or disturbed input features.

A.5.1 Stacked histograms for the basic and adversarial training

With the help of the DY+jets selection (see Ref. [48] for a more detailed description) and the CvsB discriminator, the following comparison will give a brief insight into the influence of adversarial attacks and adversarial training on the agreement between data and simulation. This particular example has been chosen, because the resulting distributions span most of the possible range between 0 and 1 and the shapes largely resemble the pre-calibration plots for DeepCSV in Ref. [48]. For the histograms, the total number of simulated samples is scaled to

match the data (normalization). In this section, the models are evaluated only at epoch 200, this corresponds to the last checkpoint during the basic and adversarial training. Without any calibration, it is expected that the shapes show some initial disagreement between simulated jets and data [48] (see the first row in Fig. A.19 for raw inputs, evaluated with the basic training (left) or adversarial training (right)). Despite the fact that the bin size used for this study is finer than the binning in Ref. [48] (by a factor of 2), there are no large fluctuations and the deviation of the ratios from 1 is limited within reasonable bounds. A small discrepancy from the overall trend can be observed both for the basic training as well as the adversarially trained model, at discriminator values of approximately 0.45 and 0.52, respectively. These are likely related to jets with many default values of the inputs. Theoretically, it is possible to end up with such excesses if the discriminator value itself is defaulted ($-1/(-1+(-1)) = 0.5$), but here this can not be the case, because the custom tagger never outputs a -1 value in the first place (unlike DeepCSV). For the custom tagger, default discriminator values only happen if there is a division by 0 (which resembles the configuration for DeepJet in Ref. [48]).

Comparing the basic and adversarial training with raw samples only, a slightly better agreement between data and simulation is observed for the adversarially trained network. This finding, which applies to most selections and discriminators and tagger outputs under investigation, will be quantified in a subsequent test of the agreement (see Section A.5.3). The next check involves randomly distorted inputs (only for simulation, using $\sigma = 0.01$), with which possible detector effects or simulation artefacts can be modelled. For the basic training (left, second row), this results in a better agreement between simulation and data, where the model is also susceptible if only this non-systematic approach is followed. No large changes are observed for the adversarially trained model when noise is applied to the simulated jets (right, second row). The third row depicts how the models react when a systematic FGSM attack is applied to simulation (the white-box FGSM attack can not be used for data, because this requires knowledge of the truth label). From the left plot (third row) it is evident that the FGSM attack with a moderate $\epsilon = 0.01$ already "overcorrects" the discriminator shape for the basic training. This network is highly vulnerable, where the ratios are basically inverted and do not move closer to 1. Surely, another magnitude of the attack would be necessary for this particular model. On the other hand, for the adversarially trained model this exact setup moves the Data/MC ratios much closer to 1. This technique is not yet optimized, but considering that some fine-tuning can be done and that the bin size could be increased to match the plots of Ref. [48], it looks like the FGSM attack on top of the adversarially trained model allows to improve the agreement between simulation and data drastically, without deriving scale factors. This is remarkable, as the attack does not know *a priori* what needs to be done in order to mitigate possible simulation artefacts, which do not occur in data and what steps are necessary to improve the agreement with data consequently (the attack only knows the loss manifolds for all flavours and the model parameters [5, 63]).

A.5.2 Scale factors

To obtain scale factors, the current setup is only configured to work with the CvsB and CvsL discriminators [48]. The custom tagger is not optimized for c-tagging and several bins of the discriminator shapes show only insufficient statistics. Although the smoothness of the distributions is improved by the reweighting and focal loss and the large peaks at 0 or 1 do not occur anymore, not the entire CvsL–CvsB phase space is covered with entries, which makes drawing conclusions difficult. When deriving scale factors, there will be a central part with small (statistical) uncertainties, but the outer part can not be used for the iterative fit (the binning and fitting procedure is described in Ref. [48]). With this limited starting position, the scale factors are subject to change when a hyperparameter tuning is conducted to improve the statistics in the entire CvsL–CvsB plane. Where the iterative fit is impossible, a scale factor of

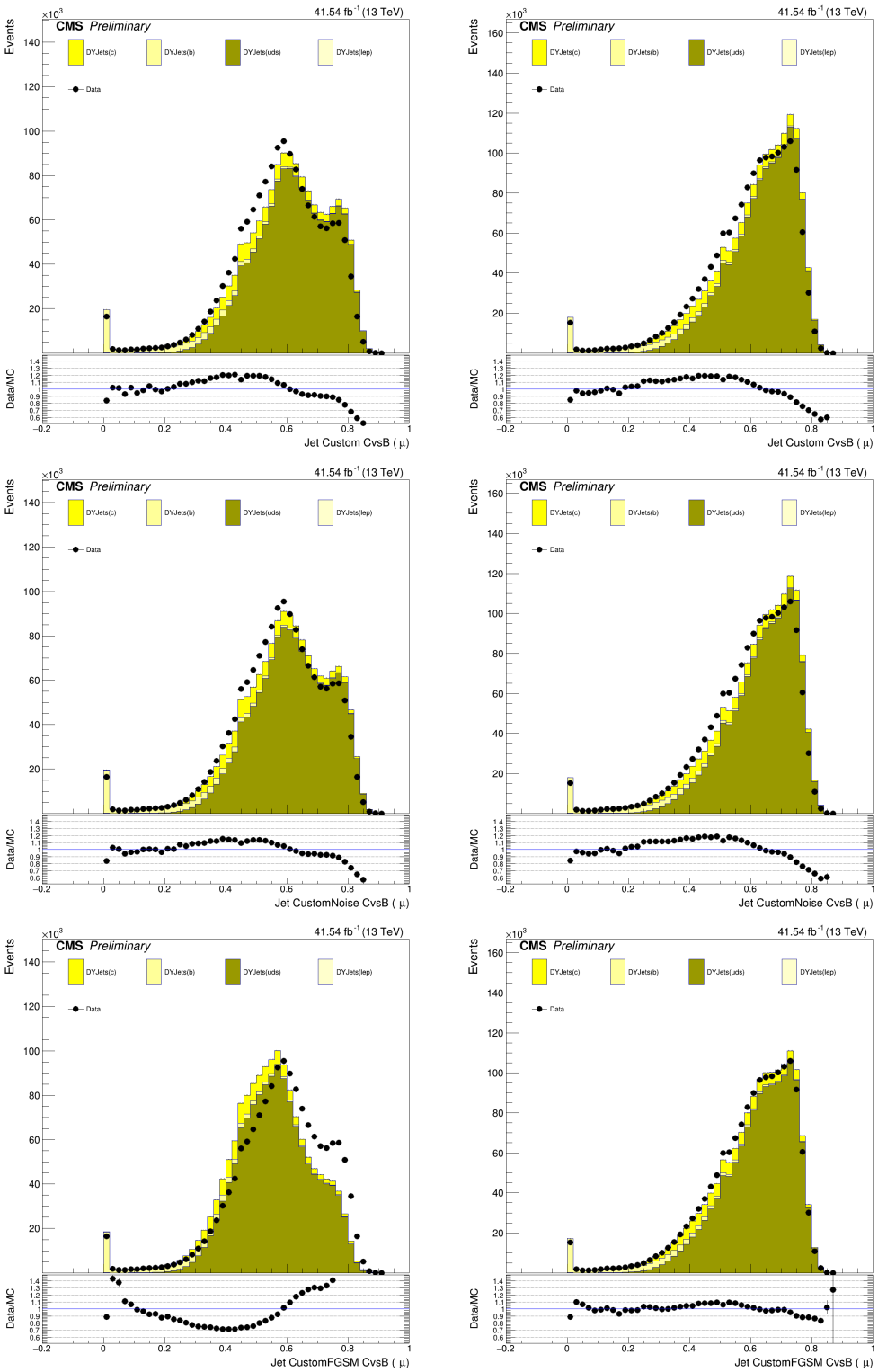

Figure A.19: Stacked histograms of the CvsB discriminator for the DY+jets selection, evaluated with the basic model (left column) and the adversarially trained model (right column). Concerning the simulated samples, the first row utilizes raw inputs, the second row results from the addition of a Gaussian noise term (with $\sigma = 0.01$) and the third row corresponds to systematically distorted inputs (FGSM attack with $\epsilon = 0.01$). The measured data distributions are identical for all plots and have not been modified by any distortion (= raw).

1 is assigned, using the value with which the scale factors are initialized [48].

In Figs. A.20 and A.21, the left column shows the derived scale factors, while the right columns display the corresponding (statistical) uncertainties. From the first to third row, there are scale factors for the light, c and b jet selections, where the CvsB binning is kept constant and the CvsL bins are allowed to float [48]. In this 2D plane, a better tagger would place light jets mostly on the left side, c jets should appear in the upper right corner and b jets would occur at the lower edge [48]. With the help of the uncertainty plots it is easy to figure out where most jets are located in the CvsL–CvsB plane, because low statistical uncertainties indicate a large number of jets in that bin. For the light and b jets selection, the adversarially trained model seems to do better, while for c jets, the basic training would have advantages, looking only at the placement of jets with undistorted inputs. Overall, there are many bins with scale factors below 1, this matches the pattern shown in Fig. A.19, where the initial Data/MC ratios in the central part of the discriminator distributions are mostly greater than 1, so the scale factors would correct this behaviour. The adversarially trained model shows a few more bins where the resulting scale factors are close to 1 (green colour), especially in those regions where the basic training does not place any jets. An example would be the upper central region for the light selection, where the uncertainties for the adversarially trained model are low and at the same time, the scale factors are close to 1, already for the non-calibrated distributions. There is practically no bin for the basic training, where the uncertainties are low (dark blue in the right column) and the scale factors are approximately 1, not so for the adversarial training, which is best observed for the light selection, but to a lesser extent also visible for the other selections. The effects are still only marginal, but it can be recorded that the adversarial training seems to move the initial agreement between data and simulation in the right direction.

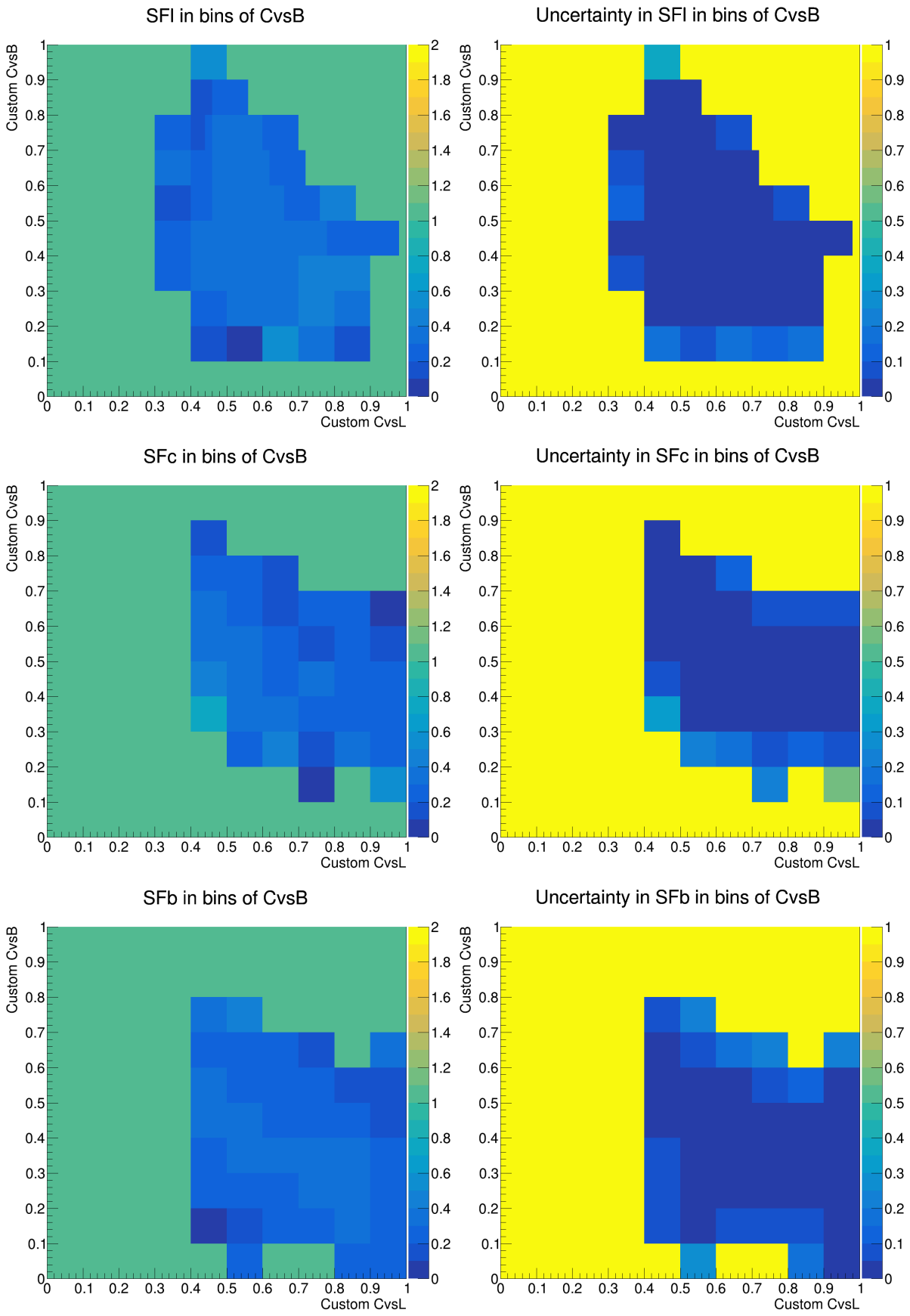

Figure A.20: CvsL / CvsB scale factors in bins of CvsB, for the basic classifier at epoch 200, evaluated on raw inputs. The scale factors are given in the left column, with corresponding uncertainties in the right column. The rows correspond to the different selections (l, c, b).

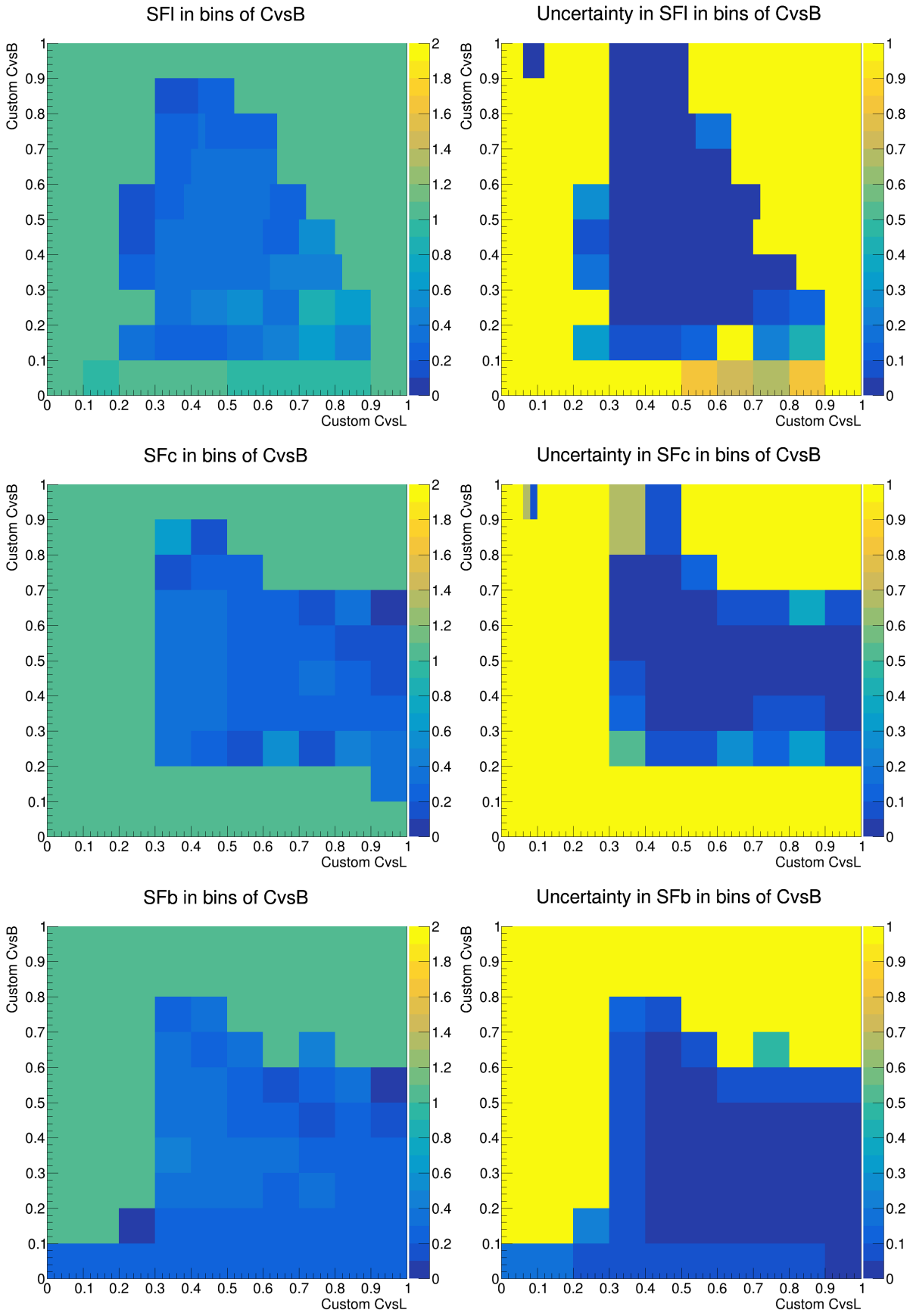

Figure A.21: CvsL / CvsB scale factors in bins of CvsB, for the adversarial training at epoch 200, evaluated on raw inputs. The scale factors are given in the left column, with corresponding uncertainties in the right column. The rows correspond to the different selections (l, c, b).

A.5.3 Agreement between data and simulation as a function of epoch

Investigating the dependence of the disagreement between simulation and data as a function of epoch is done by calculating the Kullback-Leibler (KL) divergence [110] between the respective distributions (stacked histograms). This distance measure between two distributions P and Q (here: discrete) is defined as

$$D_{\mathrm{KL}}(P \,||\, Q) = \sum_k P_k \ln \frac{P_k}{Q_k}, \tag{A.1}$$

although strictly speaking, for it to be a distance measure, the expression would have to be symmetric in P and Q [2], which is achieved by the Jensen-Shannon divergence [127]. In the expression utilized here, there is a similarity to the cross entropy, as introduced in Eq. 2.5. The case where both distributions are identical (P_k is equal to Q_k for all k) results in a KL divergence of 0, otherwise, the KL divergence is positive [2]. Here, Q is the distribution of simulated jets (summed over all flavours), P corresponds to data and k identifies a single bin of the histogram, of which there are 60. Note that the KL divergence is a summary quantity, which depends on the number of bins [125, 110]. When interpreting this quantity, there is no information about which parts of the distributions agree well. All histograms use the range between -0.2 and 1, potential default bins would occur at -0.1 [48]. To ensure that the KL divergence can be calculated also for bins with $Q_k = 0$, a tiny value is inserted as a replacement (0.00001). Due to the opposite-sign minus same-sign (OS-SS) subtraction (explanation see Ref. [48]), there could be negative values for Q_k or P_k, this is handled by setting both of them to 0, which in turn is handled as a special case by SciPy [128], namely inside the function `scipy.stats.entropy`, assigning a value of 0 for this k-th element in the sum. Therefore the KL divergences that correspond to the W+c selection are less descriptive. A second case that does not contribute to the sum is the one where P_k is zero, but Q_k is not, there the current implementation also assigns a value of 0 for this k-th bin. The conventions followed for the KL divergence are in accordance with $\lim_{x\to 0} x \ln x = 0$ [2]. Figure A.22 shows the evolution of the KL divergence between data and simulation as a function of epoch (evaluating the basic and adversarial training at epochs $1, 5, 10, 50, 100, 150$ and 200), individually for all tagger outputs and discriminators, as well as the three different selections.

These preliminary studies, which compare the basic and adversarial training, indicate that

- for most selections and tagger outputs (or discriminators) considered, the disagreement between data and simulation increases with the number of epochs (with few exceptions that are mostly constant over epoch or fluctuate),
- in most cases, the adversarially trained model shows a smaller divergence between data and simulation (main exception after 200 epochs: semileptonic $\mathrm{t\bar{t}}$ selection).

When also looking at the studies with the help of stacked histograms and scale factors, it can be summarized that

- the effect on the (dis-)agreement between data and simulation when applying adversarial attacks is more visible for the baseline classifier than for the adversarially trained network (overcorrection for the basic training, but promising agreement for adversarial training after the FGSM attack, with a better agreement on raw inputs as well),
- for the adversarial training (compared to the basic training), several bins in the 1D histograms and (2D) scale factors show initial ratios closer to one, as far as this can be studied, given the limited intervals that are actually filled with tagger scores.

These findings suggest that a more detailed investigation should follow, as the adversarial training seems to counteract the discrepancies between data and simulation to some extent. It should be noted that the training is not optimized to include the scale factors *a priori* and that the derivation only happens afterwards, hoping that the adversarial training improves

the agreement between data and simulation. Given that this is so far only a trial-and-error method, the connection between Data/MC agreement and adversarial training / adversarial attacks is already remarkably strong. It is also clear that improvements of the current setup are necessary; additionally, methods like transfer learning / domain adaption (see Refs. [120, 121]) could be used to improve the Data/MC agreement already for undisturbed inputs.

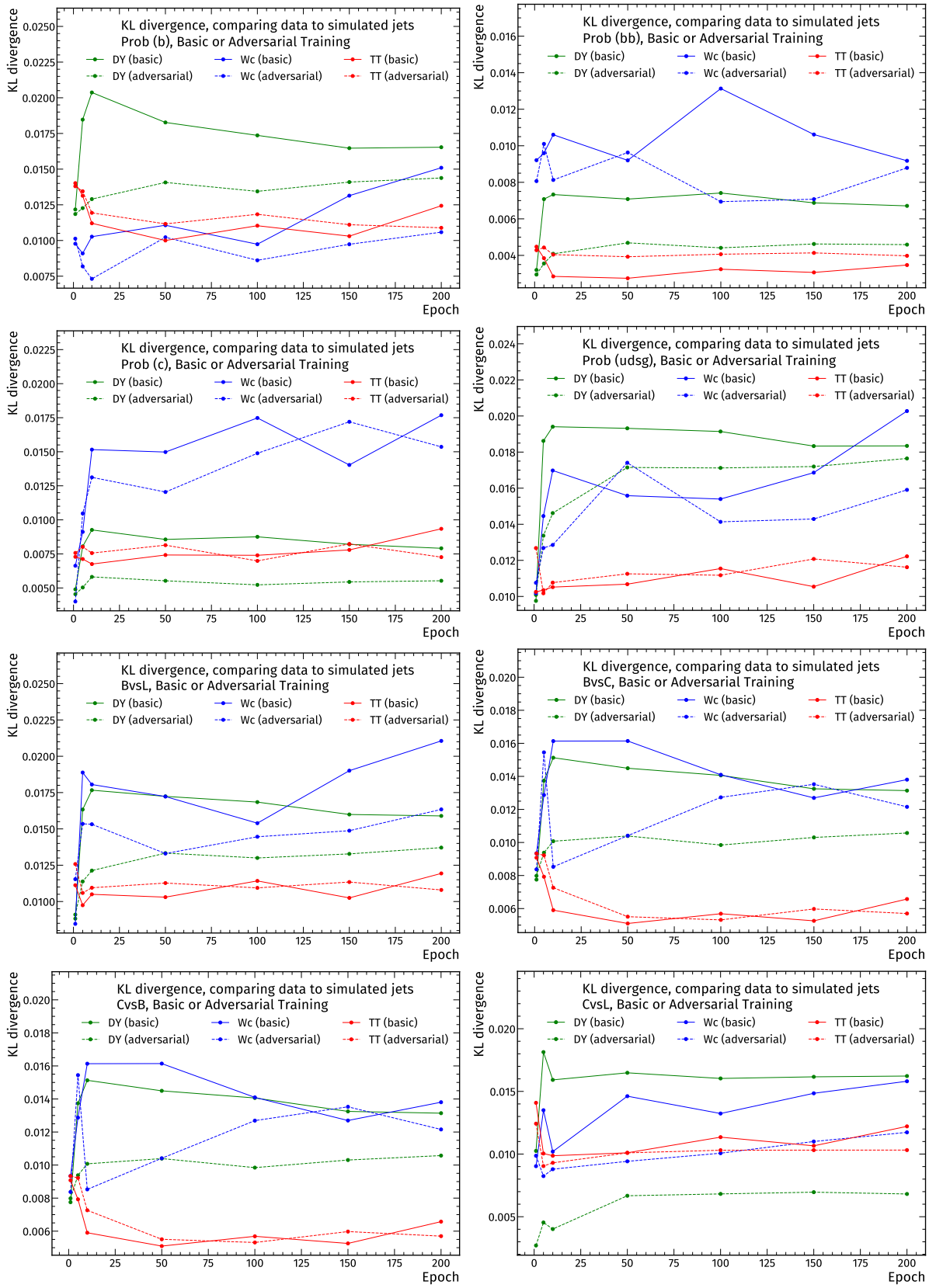

Figure A.22: KL divergences for tagger outputs and discriminators between data and simulation for a set of checkpoints of the basic training (solid lines) and adversarial training (dashed lines), split by selection. Every individual figure shows one output (b, bb, c, udsg) or discriminator (BvsL, BvsC, CvsB, CvsL) only, the colours indicate the selection (DY: green, W+c: blue, semileptonic $t\bar{t}$: red).

A.6 Utilized technologies

Table A.2: Software packages that are either used for specific tasks in python or as standalone applications.

Data preparation, analysis	
NumPy [68]	Fast, vectorized array computations.
SciPy [128]	Statistical evaluation, binning.
uproot4 [84]	File reading (`.root` format).
awkward1 [85]	Manipulations with jagged arrays.
coffea [129]	Columnar analysis, histogramming for HEP.
Model implementation, training, evaluation	
PyTorch [67]	Training and inference. Extended with code available under Refs. [98, 100, 102]
scikit-learn [95]	Splitting of the data set, evaluation with performance metrics.
Plotting, visualization	
matplotlib [130]	General plotting tool for python.
mplhep [131]	matplotlib wrapper for easy plotting required in high energy physics (HEP).
Inkscape [132]	Vectorized graphics, for all custom visualizations in Figs. 1.6,1.7,1.8,2.1,2.5,3.3,4.14

Table A.3: Collection of frameworks and projects that have contributed to the results presented in this thesis.

CMS specific	
VHcc-cTagSF (GitHub) [126]	c-tagging scale factor measurement for the resolved VHcc Analysis.
PFNano (GitHub) [86]	NanoAOD framework for testing jet algorithms.
AI safety	
advjets-mlhep2020 (GitHub) [10]	ipython notebooks, MLHEP 2020 tutorial on adversarial attacks on jets.
AI-safety (GitHub) [133]	Code used in Ref. [9] (Bachelor thesis N. Frediani).

Table A.4: Utilized services, hardware and computing sites to carry out the training and evaluation.

High-Performance	
RWTH Compute Cluster Linux (HPC) [134]	Preparation, training and evaluation of the model with Batch Systems (SLRUM). Acknowledgement: Simulations were performed with computing resources granted by RWTH Aachen University under projects `rwth0583`.
High-Throughput	
National Analysis Facility (DESY) [135]	Evaluation on data and simulation (i.e. all checks in Section A.5) with the Batch Infrastruktur Resource (BIRD, HTCondor) at Deutsches Elektronen Synchrotron (DESY), Hamburg.

Bibliography

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning", *Nat. Cell Biol.* **521** (2015), no. 7553, 436, doi:10.1038/nature14539.

[2] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning". MIT Press, 2016. http://www.deeplearningbook.org (last accessed: 12.10.2021).

[3] K. Albertsson et al., "Machine Learning in High Energy Physics Community White Paper", *J. Phys. Conf. Ser.* **1085** (2018), no. 2, 022008, doi:10.1088/1742-6596/1085/2/022008, arXiv:1807.02876.

[4] B. Nachman and C. Shimmin, "AI Safety for High Energy Physics", 2019. arXiv:1910.08606.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples", 2015. arXiv:1412.6572.

[6] C. Szegedy et al., "Intriguing properties of neural networks", 2014. arXiv:1312.6199.

[7] D. Amodei et al., "Concrete Problems in AI Safety", 2016. arXiv:1606.06565.

[8] CMS Collaboration, "Identification of heavy-flavour jets with the CMS detector in pp collisions at 13 TeV", *JINST* **13** (2018) P05011, doi:10.1088/1748-0221/13/05/p05011, arXiv:1712.07158.

[9] N. Frediani, "First studies in AI-safety for jet flavour tagging at the CMS experiment", 2020. Bachelor's Thesis, RWTH Aachen University.

[10] cshimmin / C. Shimmin, "ipython notebooks for my MLHEP 2020 tutorial on adversarial attacks on jets", 2020. https://github.com/cshimmin/advjets-mlhep2020 (last accessed: 12.10.2021).

[11] D. Guest et al., "Jet flavor classification in high-energy physics with deep neural networks", *Physical Review D* **94** (Dec, 2016) doi:10.1103/physrevd.94.1120022, arXiv:1607.08633.

[12] M. Thomson, "Modern Particle Physics". Modern Particle Physics. Cambridge University Press, 2013. ISBN 9781107034266.

[13] Cush, MissMJ, "Standard Model of Elementary Particles", Sep, 2019. Released into the public domain. Wikimedia Commons. https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg (last accessed: 12.10.2021).

[14] S. Höche, "Introduction to parton-shower event generators", in *Theoretical Advanced Study Institute in Elementary Particle Physics: Journeys Through the Precision Frontier: Amplitudes for Colliders*, pp. 235–295. 2015. arXiv:1411.4085. doi:10.1142/9789814678766_0005.

[15] P. D. Group, "Review of Particle Physics", *PTEP* **2020** (2020), no. 8, 083C01, doi:10.1093/ptep/ptaa104.

[16] T. Sjöstrand et al., "An introduction to PYTHIA 8.2", *Computer Physics Communications* **191** (Jun, 2015) 159–177, doi:10.1016/j.cpc.2015.01.024, arXiv:1410.3012.

[17] K. G. Wilson, "Confinement of quarks", *Phys. Rev. D* **10** (Oct, 1974) 2445–2459,

doi:10.1103/PhysRevD.10.2445.

[18] M. Gell-Mann, "A schematic model of baryons and mesons", *Physics Letters* **8** (1964), no. 3, 214–215, doi:10.1016/S0031-9163(64)92001-3.

[19] LHC Study Group, "The Large Hadron Collider: Conceptual design", technical report, October, 1995. CERN-AC-95-05-LHC. http://cds.cern.ch/record/291782 (last accessed: 12.10.2021).

[20] O. S. Brüning et al., "LHC Design Report". CERN Yellow Reports: Monographs. CERN, Geneva, 2004. CERN-2004-003. https://cds.cern.ch/record/782076 (last accessed: 12.10.2021). doi:10.5170/CERN-2004-003-V-1.

[21] L. R. Evans and P. Bryant, "LHC Machine", *JINST* **3** (2008) S08001. 164 p, doi:10.1088/1748-0221/3/08/S08001. This report is an abridged version of the LHC Design Report (CERN-2004-003).

[22] CERN Education, Communications and Outreach Group, "LHC Guide", Mar, 2017. CERN-Brochure-2017-002-Eng. http://cds.cern.ch/record/2255762 (last accessed: 12.10.2021).

[23] E. Mobs, "The CERN accelerator complex - 2019. Complexe des accélérateurs du CERN - 2019", Jul, 2019. CERN-GRAPHICS-2019-002. https://cds.cern.ch/record/2684277 (last accessed: 12.10.2021).

[24] CMS Collaboration, "The CMS Experiment at the CERN LHC", *JINST* **3** (2008) S08004, doi:10.1088/1748-0221/3/08/S08004.

[25] ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider", *JINST* **3** (2008) S08003, doi:10.1088/1748-0221/3/08/S08003.

[26] ALICE Collaboration, "The ALICE experiment at the CERN LHC", *JINST* **3** (2008) S08002, doi:10.1088/1748-0221/3/08/S08002.

[27] LHCb Collaboration, "The LHCb Detector at the LHC", *JINST* **3** (2008) S08005, doi:10.1088/1748-0221/3/08/S08005.

[28] ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC", *Physics Letters B* **716** (Sep, 2012) 1–29, doi:10.1016/j.physletb.2012.08.020, arXiv:1207.7214.

[29] CMS Collaboration, "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC", *Physics Letters B* **716** (Sep, 2012) 30–61, doi:10.1016/j.physletb.2012.08.021, arXiv:1207.7235.

[30] O. Aberle et al., "High-Luminosity Large Hadron Collider (HL-LHC): Technical design report". CERN Yellow Reports: Monographs. CERN, Geneva, 2020. CYRM-2020-0010. https://cds.cern.ch/record/2749422 (last accessed: 12.10.2021). doi:10.23731/CYRM-2020-0010.

[31] CMS Collaboration, "The CMS Phase-1 Pixel Detector Upgrade", *JINST* **16** (2021), no. 02, P02027, doi:10.1088/1748-0221/16/02/P02027, arXiv:2012.14304.

[32] V. Ivanchenko et al., "CMS Full Simulation for Run 3", *EPJ Web of Conferences* **251** (01, 2021) 03016, doi:10.1051/epjconf/202125103016.

[33] G. Apollinari et al., "High-Luminosity Large Hadron Collider (HL-LHC): Preliminary Design Report". CERN Yellow Reports: Monographs. CERN, Geneva, 2015. CERN-2015-005. https://cds.cern.ch/record/2116337 (last accessed: 12.10.2021). doi:10.5170/CERN-2015-005.

[34] CMS Collaboration, "Technical Proposal for the Phase-II Upgrade of the CMS Detector", technical report, Geneva, Jun, 2015. CERN-LHCC-2015-010, LHCC-P-008, CMS-TDR-15-02. http://cds.cern.ch/record/2020886 (last accessed:

12.10.2021).

[35] T. Sakuma and T. McCauley, "Detector and Event Visualization with SketchUp at the CMS Experiment", *J. Phys. Conf. Ser.* **513** (2014) 022032, doi:10.1088/1742-6596/513/2/022032, arXiv:1311.4942. Cutaway diagrams of CMS detector. https://cds.cern.ch/record/2665537 (last accessed: 12.10.2021).

[36] CMS Collaboration, "CMS Technical Design Report for the Pixel Detector Upgrade", technical report, Sep, 2012. CERN-LHCC-2012-016, CMS-TDR-11. https://cds.cern.ch/record/1481838 (last accessed: 12.10.2021).

[37] CMS Collaboration, "Description and performance of track and primary-vertex reconstruction with the CMS tracker", *Journal of Instrumentation* **9** (Oct, 2014) P10009–P10009, doi:10.1088/1748-0221/9/10/p10009, arXiv:1405.6569.

[38] CMS Collaboration, "Electron and photon reconstruction and identification with the CMS experiment at the CERN LHC", *JINST* **16** (2021), no. 05, P05014, doi:10.1088/1748-0221/16/05/P05014, arXiv:2012.06888.

[39] CMS Collaboration, "Calibration of the CMS hadron calorimeters using proton-proton collision data at $\sqrt{s} = 13$ TeV", *JINST* **15** (2020), no. 05, P05002, doi:10.1088/1748-0221/15/05/P05002, arXiv:1910.00079.

[40] CMS Collaboration, "Performance of the CMS muon detector and muon reconstruction with proton-proton collisions at $\sqrt{s} = 13$ TeV", *JINST* **13** (2018), no. 06, P06015, doi:10.1088/1748-0221/13/06/P06015, arXiv:1804.04528.

[41] CMS Collaboration, "The CMS trigger system", *JINST* **12** (2017), no. 01, P01020, doi:10.1088/1748-0221/12/01/P01020, arXiv:1609.02366.

[42] CMS Collaboration, "Particle-flow reconstruction and global event description with the CMS detector", *Journal of Instrumentation* **12** (Oct, 2017) P10003–P10003, doi:10.1088/1748-0221/12/10/p10003, arXiv:1706.04965.

[43] M. Cacciari, G. P. Salam, and G. Soyez, "The anti-k_t jet clustering algorithm", *Journal of High Energy Physics* **2008** (Apr, 2008) 063–063, doi:10.1088/1126-6708/2008/04/063, arXiv:0802.1189.

[44] G. P. Salam, "Towards jetography", *The European Physical Journal C* **67** (May, 2010) 637–686, doi:10.1140/epjc/s10052-010-1314-6, arXiv:0906.1833.

[45] CMS Collaboration, "Pileup mitigation at CMS in 13 TeV data", *Journal of Instrumentation* **15** (Sep, 2020) P09018–P09018, doi:10.1088/1748-0221/15/09/p09018, arXiv:2003.00503.

[46] CMS Collaboration, "Jet energy scale and resolution in the CMS experiment in pp collisions at 8 TeV", *Journal of Instrumentation* **12** (Feb, 2017) P02014–P02014, doi:10.1088/1748-0221/12/02/p02014, arXiv:1607.03663.

[47] R. Kogler et al., "Jet Substructure at the Large Hadron Collider: Experimental Review", *Rev. Mod. Phys.* **91** (2019), no. 4, 045003, doi:10.1103/RevModPhys.91.045003, arXiv:1803.06991.

[48] CMS Collaboration, "Calibration of charm jet identification algorithms using proton-proton collision events at $\sqrt{s} = 13$ TeV", 2021. CMS-PAS-BTV-20-001. https://cds.cern.ch/record/2758866 (last accessed: 12.10.2021).

[49] CMS Collaboration, "Observation of Higgs Boson Decay to Bottom Quarks", *Physical Review Letters* **121** (Sep, 2018) doi:10.1103/physrevlett.121.121801, arXiv:1808.08242.

[50] CMS Collaboration, "A search for the standard model Higgs boson decaying to charm quarks", *JHEP* **03** (2020) 131, doi:10.1007/JHEP03(2020)131,

arXiv:1912.01662.

[51] E. Bols et al., "Jet flavour classification using DeepJet", *Journal of Instrumentation* **15** (Dec, 2020) P12012–P12012, doi:10.1088/1748-0221/15/12/p12012, arXiv:2008.10519.

[52] E. Stevens, L. Antiga, and T. Viehmann, "Deep learning with PyTorch". Manning Publications Company, 2020. https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf (last accessed: 12.10.2021).

[53] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks* **2** (1989), no. 5, 359–366, doi:10.1016/0893-6080(89)90020-8.

[54] G. V. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems* **2** (1989) 303–314, doi:10.1007/BF02551274.

[55] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)", *IEEE Access* **6** (2018) 52138–52160, doi:10.1109/ACCESS.2018.2870052.

[56] S. Mohseni et al., "Practical Machine Learning Safety: A Survey and Primer", 2021. arXiv:2106.04823.

[57] J. Su, D. V. Vargas, and K. Sakurai, "One Pixel Attack for Fooling Deep Neural Networks", *IEEE Transactions on Evolutionary Computation* **23** (Oct, 2019) 828–841, doi:10.1109/tevc.2019.2890858, arXiv:1710.08864.

[58] C. Collard, "Commissioning of the b-tagging at 13 TeV using ak4 jets recorded in 2015", 2015. CMS-AN-15-283.

[59] CMS Collaboration, "Commissioning studies on the Phase I pixel detector of CMS in early 2017 proton-proton collisions at 13 TeV", Aug, 2017. CMS-DP-2017-037. http://cds.cern.ch/record/2281817 (last accessed: 12.10.2021).

[60] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale", 2017. arXiv:1611.01236.

[61] Fleshgrinder, "Gaussian distribution", Jun, 2009. Released into the public domain. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Gaussian_distribution.svg (last accessed: 12.10.2021).

[62] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "Robustness of classifiers: from adversarial to random noise", 2016. arXiv:1608.08967. Accepted to NIPS 2016.

[63] A. Chakraborty et al., "Adversarial Attacks and Defences: A Survey", 2018. arXiv:1810.00069.

[64] A. Madry et al., "Towards Deep Learning Models Resistant to Adversarial Attacks", 2019. arXiv:1706.06083. ICLR'18.

[65] A. Fawzi, O. Fawzi, and P. Frossard, "Analysis of classifiers' robustness to adversarial perturbations", 2016. arXiv:1502.02590.

[66] U. Shaham, Y. Yamada, and S. Negahban, "Understanding adversarial training: Increasing local stability of supervised models through robust optimization", *Neurocomputing* **307** (Sep, 2018) 195–204, doi:10.1016/j.neucom.2018.04.027, arXiv:1511.05432.

[67] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library", in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. arXiv:1912.01703. NeurIPS 2019.

[68] C. R. Harris et al., "Array programming with NumPy", *Nature* **585** (September, 2020)

357–362, doi:10.1038/s41586-020-2649-2, arXiv:2006.10256.

[69] IT Center, RWTH Aachen University, "Batch Systems (SLURM)", 2021. https://help.itc.rwth-aachen.de/service/rhr4fjjutttf/article/e018f684c5624ae6b9bf7f0994d399f2/ (last accessed: 12.10.2021).

[70] A. J. Larkoski, I. Moult, and B. Nachman, "Jet substructure at the Large Hadron Collider: A review of recent advances in theory and machine learning", *Physics Reports* **841** (Jan, 2020) 1–63, doi:10.1016/j.physrep.2019.11.001, arXiv:1709.04464.

[71] PyTorch, "CrossEntropyLoss", 2021. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html (last accessed: 12.10.2021).

[72] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels", in *32nd Conference on Neural Information Processing Systems (NeurIPS)*. 2018. arXiv:1805.07836.

[73] N. Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research* **15** (2014), no. 1, 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html (last accessed: 12.10.2021).

[74] PyTorch, "Dropout", 2021. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html (last accessed: 12.10.2021).

[75] G. E. Hinton et al., "Improving neural networks by preventing co-adaptation of feature detectors", 2012. arXiv:1207.0580.

[76] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", in *3rd International Conference for Learning Representations (ICLR)*. 2015. arXiv:1412.6980.

[77] S. Ruder, "An overview of gradient descent optimization algorithms", 2017. arXiv:1609.04747.

[78] PyTorch, "Adam", 2021. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam (last accessed: 12.10.2021).

[79] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search", in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, pp. 3–12. 1992. doi:10.1109/NNSP.1992.253713.

[80] J. Alwall et al., "MadGraph 5 : Going Beyond", *JHEP* **06** (2011) 128, doi:10.1007/JHEP06(2011)128, arXiv:1106.0522.

[81] S. Frixione, P. Nason, and C. Oleari, "Matching NLO QCD computations with Parton Shower simulations: the POWHEG method", *JHEP* **11** (2007) 070, doi:10.1088/1126-6708/2007/11/070, arXiv:0709.2092.

[82] S. Agostinelli et al., "GEANT4—a simulation toolkit", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **506** (2003), no. 3, 250–303, doi:10.1016/S0168-9002(03)01368-8.

[83] CMS Collaboration, "Identification of b-quark jets with the CMS experiment", *Journal of Instrumentation* **8** (Apr, 2013) P04013–P04013, doi:10.1088/1748-0221/8/04/p04013, arXiv:1211.4462.

[84] J. Pivarski et al., "scikit-hep/uproot4: 0.1.2", *Zenodo* (2020) doi:10.5281/zenodo.4340632.

[85] J. Pivarski et al., "scikit-hep/awkward-1.0: 0.4.5", *Zenodo* (2020) doi:10.5281/zenodo.4341376.

[86] cms-jet / CMS Collaboration, "PFNano", 2021. GitHub repository.

https://github.com/cms-jet/PFNano (last accessed: 12.10.2021).

[87] CMS Collaboration, "The CMS NanoAOD data tier", 2021. https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookNanoAOD (last accessed: 12.10.2021).

[88] CMS Collaboration, "Jet Flavour Identification (MC Truth)", 2016. https://twiki.cern.ch/twiki/bin/viewauth/CMSPublic/SWGuideBTagMCTools (last accessed: 12.10.2021).

[89] G. Petrucciani, A. Rizzi, and C. Vuosalo, "Mini-AOD: A New Analysis Data Format for CMS", *Journal of Physics: Conference Series* **664** (Dec, 2015) 072052, doi:10.1088/1742-6596/664/7/072052, arXiv:1702.04685.

[90] M. Cacciari and G. P. Salam, "Pileup subtraction using jet areas", *Phys. Lett. B* **659** (2008) 119–126, doi:10.1016/j.physletb.2007.09.077, arXiv:0707.1378.

[91] S. Moortgat, M. Verzetti, G. V. Onsem, and K. Kovitanggoon, "Training and validation of a new c-tagging algorithm in TMVA", 2013. CMS-AN-16-052.

[92] Nikolas Frediani. Private conversation, 2021.

[93] M. Kuhn and K. Johnson, "Feature Engineering and Selection: A Practical Approach for Predictive Models". Chapman & Hall/CRC Data Science Series. CRC Press, 2019. https://bookdown.org/max/FES (last accessed: 12.10.2021). ISBN 9781351609463.

[94] P. V. Mulders et al., "Implementation and training of the Combined Secondary Vertex MVA b-tagging algorithm in CMSSW", 2013. CMS-AN-12-441.

[95] F. Pedregosa et al., "Scikit-learn: Machine learning in python", *Journal of Machine Learning Research* **12** (01, 2012) arXiv:1201.0490. https://jmlr.org/papers/v12/pedregosa11a.html (last accessed: 12.10.2021).

[96] M. Galar et al., "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **42** (2012), no. 4, 463–484, doi:10.1109/TSMCC.2011.2161285.

[97] P. Branco, L. Torgo, and R. P. Ribeiro, "A Survey of Predictive Modeling on Imbalanced Domains", *ACM Comput. Surv.* **49** (August, 2016) doi:10.1145/2907070, arXiv:1505.01658.

[98] ufoym / M. Yang, "Imbalanced Dataset Sampler", 2021. GitHub repository. https://github.com/ufoym/imbalanced-dataset-sampler (last accessed: 12.10.2021).

[99] T.-Y. Lin et al., "Focal Loss for Dense Object Detection", in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007. 2017. arXiv:1708.02002. doi:10.1109/ICCV.2017.324.

[100] AdeelH / A. Hassan, "Multi-class Focal Loss", 2021. GitHub repository. https://github.com/AdeelH/pytorch-multi-class-focal-loss (last accessed: 12.10.2021).

[101] H. Carlens, "Better Data Loading: 20x PyTorch Speed-Up for Tabular Data", 2020. https://towardsdatascience.com/better-data-loading-20x-pytorch-speed-up-for-tabular-data-e264b9e34352 (last accessed: 12.10.2021).

[102] hcarlens / H. Carlens, "pytorch-tabular", 2020. GitHub repository. https://github.com/hcarlens/pytorch-tabular (last accessed: 12.10.2021).

[103] PyTorch, "Source code for torch.utils.data.dataloader", 2021.

https://pytorch.org/docs/stable/_modules/torch/utils/data/dataloader.html#DataLoader (last accessed: 12.10.2021).

[104] A. Cuyt, B. Verdonk, S. Becuwe, and P. Kuterna, "A Remarkable Example of Catastrophic Cancellation Unraveled", *Computing* **66** (05, 2001) 309–320, doi:10.1007/s006070170028.

[105] D. Goldberg, "What every computer scientist should know about floating-point arithmetic", *ACM computing surveys (CSUR)* **23** (1991), no. 1, 5–48, doi:10.1145/103162.103163.

[106] J. Davis and M. Goadrich, "The Relationship between Precision-Recall and ROC Curves", in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, p. 233–240. Association for Computing Machinery, New York, NY, USA, 2006. doi:10.1145/1143844.1143874.

[107] D. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation", *Mach. Learn. Technol.* **2** (01, 2008) doi:10.9735/2229-3981, arXiv:2010.16061.

[108] G. Pólya, "Über den zentralen Grenzwertsatz der Wahrscheinlichkeitsrechnung und das Momentenproblem", *Mathematische Zeitschrift* **8** (1920) 171–181, doi:10.1007/bf01206525.

[109] A. Ghosh, B. Nachman, and D. Whiteson, "Uncertainty-aware machine learning for high energy physics", *Physical Review D* **104** (Sep, 2021) doi:10.1103/physrevd.104.056026, arXiv:2105.08742.

[110] S. Kullback and R. A. Leibler, "On Information and Sufficiency", *Ann. Math. Statist.* **22** (03, 1951) 79–86, doi:10.1214/aoms/1177729694.

[111] D. Stutz, M. Hein, and B. Schiele, "Disentangling Adversarial Robustness and Generalization", *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019) 6969–6980, doi:{10.1109/CVPR.2019.00714}, arXiv:1812.00740.

[112] H. Li et al., "Visualizing the Loss Landscape of Neural Nets", 2018. arXiv:1712.09913. NIPS 2018.

[113] S. Fort, H. Hu, and B. Lakshminarayanan, "Deep Ensembles: A Loss Landscape Perspective", 2020. arXiv:1912.02757.

[114] I. Goodfellow et al., "Generative Adversarial Networks", *Commun. ACM* **63** (October, 2020) 139–144, doi:10.1145/3422622.

[115] F. Tramèr et al., "Ensemble Adversarial Training: Attacks and Defenses", 2020. arXiv:1705.07204. International Conference on Learning Representations (ICLR) 2018.

[116] N. Papernot et al., "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks", in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597. 2016. arXiv:1511.04508. doi:10.1109/SP.2016.41.

[117] N. Papernot and P. McDaniel, "Extending Defensive Distillation", 2017. arXiv:1705.05264.

[118] S. Ni, J. Li, and H.-Y. Kao, "DropAttack: A Masked Weight Adversarial Training Method to Improve Generalization of Neural Networks", 2021. arXiv:2108.12805.

[119] J. Mok, B. Na, H. Choe, and S. Yoon, "AdvRush: Searching for Adversarially Robust Neural Architectures", 2021. arXiv:2108.01289. International conference on machine learning 2021.

[120] Y. Ganin and V. Lempitsky, "Unsupervised Domain Adaptation by Backpropagation",

2015. arXiv:1409.7495.

[121] CMS Collaboration, "A deep neural network to search for new long-lived particles decaying to jets", *Machine Learning: Science and Technology* **1** (Aug, 2020) 035012, doi:10.1088/2632-2153/ab9023, arXiv:1912.12238.

[122] A. G. Baydin et al., "Differentiable Programming in High-Energy Physics", 2020. In Snowmass 2021 Letters of Interest (LOI), Division of Particles and Fields (DPF), American Physical Society, https://www.snowmass21.org/docs/files/summaries/CompF/SNOWMASS21-CompF5_CompF3_Gordon_Watts-046.pdf.

[123] P. de Castro and T. Dorigo, "INFERNO: Inference-Aware Neural Optimisation", *Computer Physics Communications* **244** (Nov, 2019) 170–179, doi:10.1016/j.cpc.2019.06.007, arXiv:1806.04743.

[124] Institute of Electrical and Electronics Engineers - IEEE, "IEEE Standard for Floating-Point Arithmetic", *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019) 1–84, doi:10.1109/IEEESTD.2019.8766229.

[125] K. H. Knuth, "Optimal Data-Based Binning for Histograms", 2006. arXiv:physics/0605197.

[126] mondalspandan / S. Mondal, "VHcc - cTag SF", 2021. GitHub repository. https://github.com/mondalspandan/VHcc-cTagSF (last accessed: 12.10.2021).

[127] J. Lin, "Divergence measures based on the Shannon entropy", *IEEE Transactions on Information Theory* **37** (1991), no. 1, 145–151, doi:10.1109/18.61115.

[128] P. Virtanen et al., "SciPy 1.0: fundamental algorithms for scientific computing in Python", *Nature Methods* **17** (Feb, 2020) 261–272, doi:10.1038/s41592-019-0686-2, arXiv:1907.10121.

[129] L. Gray et al., "CoffeaTeam/coffea: Release v0.6.46", *Zenodo* (2020) doi:10.5281/zenodo.3266454.

[130] J. D. Hunter, "Matplotlib: A 2D graphics environment", *Computing in Science & Engineering* **9** (2007), no. 3, 90–95, doi:10.1109/MCSE.2007.55.

[131] A. Novak et al., "scikit-hep/mplhep: v.0.2.8", *Zenodo* (2020) doi:10.5281/zenodo.3766157.

[132] T. Gould et al., "Inkscape", 2021. Inkscape is a free and open-source vector graphics editor. https://inkscape.org/ (last accessed: 12.10.2021).

[133] nfredian / N. Frediani, "AI-safety", 2020. GitHub repository. https://github.com/nfredian/AI-safety (last accessed: 12.10.2021).

[134] IT Center, RWTH Aachen University, "Hochleistungsrechnen", 2021. https://www.itc.rwth-aachen.de/cms/IT-Center/Forschung-Projekte/~eubj/High-Performance-Computing/ (last accessed: 12.10.2021).

[135] Beyer, Christoph et al., "Beyond HEP: Photon and accelerator science computing infrastructure at DESY", *EPJ Web Conf.* **245** (2020) 07036, doi:10.1051/epjconf/202024507036.

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank Prof. Dr. Alexander Schmidt for giving me the opportunity to work on this exciting topic, for always being optimistic and giving encouraging feedback.

Also, I thank Prof. Dr. Martin Erdmann, for kindly agreeing to be the second examiner.

I would like to thank Andrzej Novak for supervising me, for answering my questions and always offering practical instructions and support, especially concerning the technical details when I started from scratch. I would also like to thank Xavier Coubez, who supervised me and introduced me to the topic. He always provided interesting ideas and guided me with useful examples to get deeper insights. Also, thanks go to both of them for proofreading this thesis and giving helpful comments. Further, I would like to thank Spandan Mondal. With his instructions and detailed answers to many questions, we were able to adapt the scale factor framework to work with a custom tagger. Also, I appreciate the tips and expertise that Andrey Pozdnyakov shared with me on how to submit condor jobs and how to work with the DESY environment. Many thanks also go to Nikolas Frediani, being the first to work on this specific topic. He provided example code and did important follow-up studies that were of great help for me.

I would especially like to express my gratitude to the whole working group. Despite the pandemic situation, with supervision only being possible online, everyone offered a friendly and motivating environment which contributed to the completion of this thesis. Thank you for this enjoyable experience during the past year. I'm looking forward to what lies ahead!

Also, I would like to thank the Department of Physics for facilitating the change of the study program, simply offering solutions and giving additional permissions where necessary.

Finally, special thanks go to my family, for their continuous support, understanding and encouragement when I decided to follow a different path.

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Stein, Annika — Name, Vorname/Last Name, First Name

407142 — Matrikelnummer (freiwillige Angabe) / Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit/Bachelorarbeit/~~ Masterarbeit* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present ~~paper/Bachelor thesis/~~Master thesis* entitled

Investigation of robustness of b-Tagging algorithms for the CMS Experiment

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date — Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date — Unterschrift/Signature