Master's Thesis in Physics

Tau neutrino identification with Graph Neural Networks in KM3NeT/ORCA

Submitted by

Lukas Hennig

June 1, 2023

Erlangen Centre for Astroparticle Physics

Department of Physics

Friedrich-Alexander-Universität Erlangen-Nürnberg

First reviewer: PD Dr. Thomas Eberl

Second reviewer: Prof. Dr. Uli Katz

Abstract

Neutrinos are included in the Standard Model of particle physics as electrically neutral fermions appearing in three different flavors. The experimental verification of neutrino oscillations showed that neutrinos generated in one flavor can be measured in a different flavor with a nonzero probability. The probabilities for neutrino oscillations are determined by a set of parameters that must be constrained by experiments. Measurements of the oscillation probabilities for a flavor transition into the tau flavor can contribute significantly to constraining these parameters. The KM3NeT/ORCA detector is a neutrino detector currently being constructed in the Mediterranean deep sea. It focuses on measurements of the atmospheric neutrino flux. The data of KM3NeT/ORCA can be used to find neutrino interactions induced by tau neutrinos. Since GeV energy tau neutrinos are not produced in the atmosphere, these tau interactions result from flavor oscillations. Graph Neural Networks (GNNs) are artificial neural networks using graphs as input. Since the data of KM3NeT/ORCA can be represented as graphs, this type of neural network is an excellent candidate for performing reconstruction tasks. This thesis trains GNNs to directly identify tau neutrino interactions with KM3NeT/ORCA, a task referred to as "tau identification". Several optimization steps to improve the performance of the GNNs were carried out. The composition of the available training data was optimized by generating simulated datasets with a large number of tau neutrino interactions. Furthermore, an optimized training dataset configuration was found by systematic variations of the properties of the atmospheric neutrino flux and the tau neutrino fraction. The parameters configuring the machine learning algorithm, commonly called hyperparameters, were systematically studied in an automatic hyperparameter optimization (AHPO) using the software library Ray Tune, which is the first AHPO performed in the KM3NeT Collaboration. The AHPO resulted in a training of over 2000 GNNs using a computing time budget of 46 000 GPU-hours granted to this work by the NHR@FAU. A performance analysis of the trained networks resulted in a confirmation of the hyperparameter choices made in previous studies in KM3NeT. The GNN showing the best performance in tau identification was evaluated, demonstrating its ability to increase the fraction of tau neutrino interactions in simulated data by a factor of eight to about 20% while still retaining 30% of all tau neutrino interactions.

Contents

1. Neutrino physics

The existence of neutrinos was first theorized by W. Pauli in 1930, who proposed the idea of an undiscovered electrically neutral particle in an, in his own words, "desperate attempt" to rescue the seemingly violated energy conservation in the beta decay[1]. A few years later, in 1934, E. Fermi included the neutrino in his theory of beta decay, now known as the Fermi theory[2]. It took over twenty more years until the neutrino was finally discovered by C. L. Cowan and F. Reines in their famous reactor experiment[3], paving the way for further milestones in the field of neutrino physics over the following decades. In this chapter, the current status of neutrino physics is reviewed by giving an overview of the general properties of the neutrino, followed by a more detailed discussion of the phenomenon of neutrino oscillations. After that, the still open question of whether the PMNS matrix is violating unitarity is discussed, to which this thesis can be seen as a contribution in an early stage. The chapter closes with a review of so-called atmospheric neutrinos, the most relevant class of neutrinos for this particular work.

1.1. Properties of the neutrino

The neutrino is included as a lepton in the "Standard Model of particle physics" (SM), the currently most established theory of the constituents of our universe. An overview of the particles in the SM can be seen in fig. 1.1. Neutrinos neither carry electromagnetic nor color charge, meaning they do not participate in the electromagnetic and strong interaction. The only way a neutrino can interact with matter is by the weak interaction, which is mediated by the neutral Z^0 and the charged $W^\pm$ bosons. The gravitational interaction, by which neutrinos can also interact, is left out here because it is not a part of the SM. Analogous to the charged leptons, the neutrinos appear in three different generations, also often called "flavors", typically denoted by ν_e, ν_μ, and ν_τ. As with all fermions in the SM, each neutrino flavor additionally has an antiparticle, which is

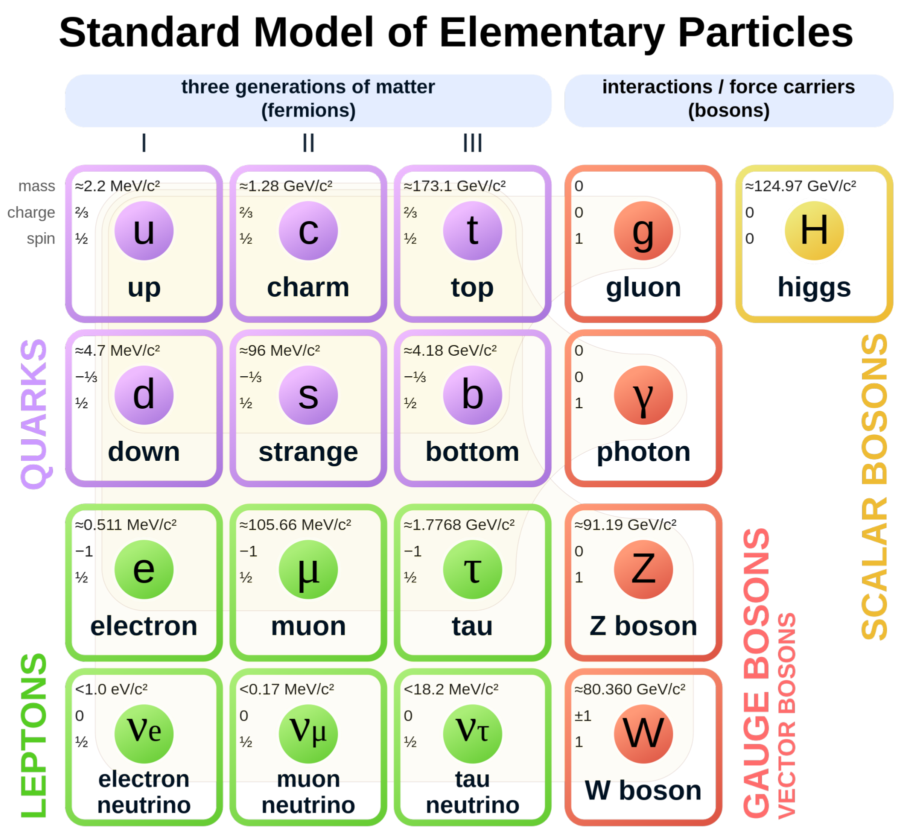


Figure 1.1.: The particles in the Standard Model. The antiparticles of the matter particles are not shown. The brown loops indicate which gauge boson interacts with which matter particles, showing that the neutrino only interacts via the weak interaction. From [4].

denoted by $\bar{\nu}_e$, $\bar{\nu}_\mu$, and $\bar{\nu}_\tau$.

Neutrinos are included as massless particles in the SM, but as can already be guessed from the upper mass bounds given in fig. 1.1, this is not the case. The phenomenon of "neutrino oscillations", which states that neutrinos can change their flavor during their propagation through space, shows that neutrinos must have a nonzero mass. Neutrino oscillations are the first experimentally verified phenomenon contradicting the Standard Model, raising many questions about the fundamental nature of the neutrino that are still not understood.

1.2. Neutrino oscillations

Around 25 years ago, in 1998, the Super-Kamiokande Collaboration published the first experimental evidence for neutrino oscillations by analyzing the atmospheric neutrino flux[5], in which they showed that a significant amount of muon neutrinos oscillated into tau neutrinos. Shortly after that, in 2002, the SNO Collaboration, using the neutrino flux emitted from the sun, reported additional evidence for the phenomenon of neutrino oscillations[6], which resolved the solar neutrino problem that was troubling physicists since the 1960s by showing that a fraction of electron neutrinos oscillated into muon neutrinos. In 2015, these significant breakthroughs were honored by the Nobel Prize awarded to Takaaki Kajita from the Super-Kamiokande Collaboration and Arthur B. McDonald from the SNO Collaboration "for the discovery of neutrino oscillations, which shows that neutrinos have mass"[7].

In the currently most established model of neutrino oscillations, a neutrino state $|\nu\rangle$ is described as a superposition of three basis vectors. There exist two bases that are interesting in this picture. The first is called the flavor basis, consisting of the basis vectors $|\nu_e\rangle$, $|\nu_\mu\rangle$, and $|\nu_\tau\rangle$. A general superposition in this basis has the form

$$|\nu\rangle = \sum_{\alpha=e,\mu,\tau} c_\alpha \, |\nu_\alpha\rangle \,, \tag{1.1}$$

where the coefficients c_α are complex numbers satisfying the normalization condition $\sum_\alpha |c_\alpha|^2 = 1$. The flavor basis vectors correspond to the neutrino states in the Standard Model depicted in fig. 1.1. A neutrino produced during a weak interaction process, e.g., a beta decay, is always produced as a flavor basis state. Analogously, a neutrino measured by a weak interaction process, e.g., an inverse beta decay, will always be measured in a flavor basis state. The phenomenon of neutrino oscillation allows a neutrino to change its flavor, meaning that the flavor $|\nu_\alpha\rangle$ in which the neutrino was produced can differ from the flavor $|\nu_\beta\rangle$ in which the neutrino was measured at a later point in time. If the states of the flavor basis would be eigenstates of the Hamiltonian H_0 that determines the propagation of neutrinos through the vacuum, a neutrino produced in flavor state $|\nu_\alpha\rangle$ would remain in this state at later points in time. Since this contradicts the experimental evidence, it can be concluded that the flavor basis states are not eigenstates of H_0. Hence, there must be another interesting basis consisting of the eigenstates of H_0. This basis is called mass basis from now on, and its basis vectors will be denoted with Latin indices $|\nu_i\rangle$ with $i = 1, 2, 3$, in contrast to the flavor eigenbasis, which always has Greek indices.

Since both the flavor basis and the mass basis describe the same Hilbert space, the states of the flavor basis must be expressable as complex linear combinations of the mass basis and vice versa. This can be denoted as

$$\begin{pmatrix} |\nu_e\rangle \\ |\nu_\mu\rangle \\ |\nu_\tau\rangle \end{pmatrix} = \begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{pmatrix} \begin{pmatrix} |\nu_1\rangle \\ |\nu_2\rangle \\ |\nu_3\rangle \end{pmatrix} . \tag{1.2}$$

The complex 3×3 matrix is commonly named the PMNS matrix and denoted as U_{PMNS}, where PMNS stands for Pontecorvo–Maki–Nakagawa–Sakata, who made fundamental contributions to the theory of neutrino oscillations [8, 9]. The PMNS matrix is typically assumed to be unitary, which implies that the total neutrino flux over all three flavors is always conserved. It can be parametrized by three mixing angles θ_{12}, θ_{23}, and θ_{13}, and an additional CP-violating phase δ, as three rotations

$$U_{\mathrm{PMNS}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & e^{-i\delta} s_{13} \\ 0 & 1 & 0 \\ -e^{i\delta} s_{13} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix} , \tag{1.3}$$

where c_{ij} is short for $\cos\theta_{\mathrm{ij}}$ and s_{ij} for $\sin\theta_{\mathrm{ij}}$. The values of these parameters are not calculatable from theory. Instead, they have to be measured by experiment. There exist two additional parameters in the PMNS matrix that are nonzero only if the neutrino turns out to be its own antiparticle, i.e., if the neutrino is a Majorana particle. However, these parameters do not influence neutrino oscillations, so that they can be ignored in this discussion.

The parameters of the PMNS matrix are measured by oscillation experiments, in which the probability of one or more flavor transitions is measured. At first, one has to derive a formula that describes how the PMNS matrix elements influence the oscillation probabilities. For this, one can start by looking at the time evolution of the mass eigenstate $|\nu_i(t)\rangle$, which is given by the Schrödinger equation

$$i\frac{\partial}{\partial t} |\nu_i(t)\rangle = E_i |\nu_i(t)\rangle , \tag{1.4}$$

where E_i is the energy of the mass eigenstate $|\nu_i\rangle$. The solutions to this equation are the plane waves

$$|\nu_i(t)\rangle = e^{-iE_i t} |\nu_i(0)\rangle , \quad E_i = \sqrt{m_i^2 + p^2}, \tag{1.5}$$

where m_i is the mass of the mass eigenstate i, and p is the momentum. In all practical cases, neutrinos can be assumed to travel approximately at the speed of light. This assumption justifies the use of the ultra-relativistic neutrino approximation, which allows the substitutions

$$p \approx E, \quad E_i \approx E + \frac{m_i^2}{2E}, \quad t \approx L, \tag{1.6}$$

where L is the distance that the neutrino propagated. These substitutions modify eq. (1.5) to the form

$$|\nu_i(L)\rangle = \exp\left(-i\frac{m_i^2 L}{2E}\right)|\nu_i\rangle \,. \tag{1.7}$$

The term in the exponent proportional to E was removed because it only leads to a global phase not affecting the physically relevant quantum mechanical probabilities. From this, one can obtain the evolution of a flavor state during its propagation by a basis transformation

$$\begin{aligned} |\nu_\alpha(L,E)\rangle &= \sum_{i=1}^{3} \langle \nu_i(L)|\nu_\alpha(L)\rangle\, |\nu_i(L)\rangle = \sum_{i=1}^{3} U_{\alpha i} \exp\left(-i\frac{m_i^2 L}{2E}\right)|\nu_i\rangle & (1.8)\\ &= \sum_{i=1}^{3} \sum_{\beta=e,\mu,\tau} U_{\alpha i} \exp\left(-i\frac{m_i^2 L}{2E}\right) U^*_{\beta i}\, |\nu_\beta\rangle \,, & (1.9)\end{aligned}$$

where U^* is the complex conjugate of U. The second equality is obtained using eq. (1.7) and the third using eq. (1.2). The probability $P_{\alpha\to\beta}(L,E)$ that a neutrino created in flavor state $|\nu_\alpha\rangle$ is measured in flavor state $|\nu_\beta\rangle$ after propagating a distance L can be obtained by taking the absolute square of the projection on $|\nu_\beta\rangle$, which yields

$$\begin{aligned} P_{\alpha\to\beta}(L,E) &= |\langle \nu_\beta|\nu_\alpha(L,E)\rangle|^2 & (1.10)\\ &= \sum_{i,j} U_{\alpha i} U^*_{\beta i} U^*_{\alpha j} U_{\beta j} \exp\left(-i\frac{\Delta m_{ij}^2 L}{2E}\right), & (1.11)\end{aligned}$$

where $\Delta m_{ij}^2 = m_i^2 - m_j^2$ is the squared mass difference. It can be seen from eq. (1.11) that the quantity L/E leads to oscillatory terms with a wavelength determined by $\Delta m_{i,j}^2$, which explains the origin of the term neutrino oscillations. In contrast, the PMNS matrix elements determine the amplitudes of the oscillations. Experiments measuring the transition probabilities as a function of L/E are therefore sensitive to the PMNS matrix elements and the squared mass differences but not, e.g., to the absolute masses of the mass eigenstates.

NuFIT 5.2 (2022)

$$|U|_{3\sigma}^{\text{w/o SK-atm}} = \begin{pmatrix} 0.803 \to 0.845 & 0.514 \to 0.578 & 0.142 \to 0.155 \\ 0.233 \to 0.505 & 0.460 \to 0.693 & 0.630 \to 0.779 \\ 0.262 \to 0.525 & 0.473 \to 0.702 & 0.610 \to 0.762 \end{pmatrix}$$

$$|U|_{3\sigma}^{\text{with SK-atm}} = \begin{pmatrix} 0.803 \to 0.845 & 0.514 \to 0.578 & 0.143 \to 0.155 \\ 0.244 \to 0.498 & 0.502 \to 0.693 & 0.632 \to 0.768 \\ 0.272 \to 0.517 & 0.473 \to 0.672 & 0.623 \to 0.761 \end{pmatrix}$$

Figure 1.2.: The 3σ confidence intervals of the absolute values of the PMNS matrix elements are shown. The top matrix shows the confidence intervals calculated without the atmospheric neutrino data from Super-Kamiokande, while the bottom matrix includes them. Unitarity is assumed for the calculation of these confidence intervals. From [10, 11].

1.3. Unitarity of the PMNS matrix

As mentioned briefly in the last section, the PMNS matrix is typically assumed to be unitary. The physical implication of this assumption is that although the theory allows changes from one flavor to another, the sum of all neutrinos over all three flavors must be conserved. This assumption is justified if there are only three neutrino flavors. However, there exist theories beyond the Standard Model that postulate the existence of further neutrino flavors called “sterile neutrinos” that do not interact via the weak interaction. In this case, the PMNS matrix defined in eq. (1.2) would only be a 3×3 submatrix of a larger $n \times n$ matrix. While the whole $n \times n$ matrix must be unitary in this case, the 3×3 submatrix would not have this constraint. Hence, if the currently ongoing experiments to constrain the PMNS matrix elements can confirm that the PMNS matrix is not unitary, this would be direct evidence for the existence of further neutrino flavors. On the other hand, if the measurements do not indicate a violation of unitarity, there is little room left for theories postulating sterile neutrinos.

The PMNS matrix elements have to be constrained sufficiently well to test the unitarity of the PMNS matrix. The current status of this task can be seen in fig. 1.2, where the 3σ confidence intervals of the absolute values of the PMNS matrix can be seen. As explained in [10, 11], the confidence levels were calculated using available data collected by many different neutrino experiments. Figure 1.2 shows that the first row of the PMNS matrix is constrained rather well compared to the second and third rows. These last two rows

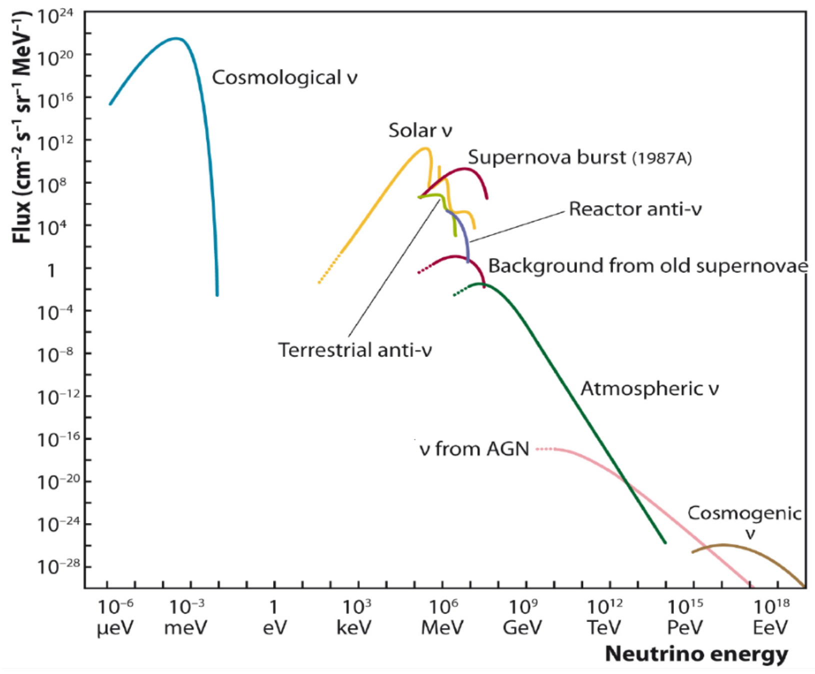


Figure 1.3.: An overview of different neutrino sources, their respective energy ranges, and their fluxes. AGN is short for "active galactic nuclei". From [12].

show up in eq. (1.11) if one searches, e.g., for the transition $\nu_\mu \to \nu_\tau$. This transition can be probed very well with atmospheric neutrino experiments like KM3NeT/ORCA, which will be discussed later in chapter 2.

1.4. Atmospheric neutrinos

As shown in eq. (1.11), the quantity L/E determines the phase of its oscillation. Luckily, many different neutrino sources exist which cover a wide range of distances L and neutrino energies E. An overview of these sources can be seen in fig. 1.3. For this work, the atmospheric neutrinos are the most important neutrino class.

Atmospheric neutrinos are produced in the Earth's atmosphere when cosmic rays originating from space impinge on the nuclei in Earth's atmosphere at a typical height of 15 km[13]. These cosmic rays are mainly composed of protons with energies of GeV nucleon^{-1}, with about 5% helium nuclei and a small fraction of heavier nuclei. As a result of the interactions between the cosmic rays with the atmosphere, Mesons are produced. Most of these mesons are unstable $\pi^{\pm}$ mesons that decay via the channels

$$\pi^{+} \rightarrow \mu^{+} + \nu_{\mu} \tag{1.12}$$

$$\pi^{-} \rightarrow \mu^{-} + \bar{\nu}_{\mu}, \tag{1.13}$$

where one muon (anti)neutrino is produced each. The emitted $\mu^{\pm}$ leptons are again unstable, decaying via the muon decay

$$\mu^{+} \rightarrow e^{+} + \bar{\nu}_{\mu} + \nu_{e} \tag{1.14}$$

$$\mu^{-} \rightarrow e^{-} + \nu_{\mu} + \bar{\nu}_{e}, \tag{1.15}$$

where one muon (anti)neutrino and one electron (anti)neutrino is produced per interaction. This interaction chain is visualized in fig. 1.4. If one looks at the neutrino products from these channels, one would expect a flavor ratio $(\nu_{\mu} + \bar{\nu}_{\mu}) / (\nu_{e} + \bar{\nu}_{e}) \approx 2$ for atmospheric neutrinos, which is indeed the expectation at energies around 1 GeV[13]. The ratio increases for higher energies because more and more muons can reach the Earth before decaying. The atmospheric neutrino flux contains practically no tau neutrinos since producing a heavy tau lepton with a rest mass of approximately $1.78\,\mathrm{GeV}\,c^{-2}$ requires too much energy.

If one fixes an arbitrary position on Earth, e.g., the position of a neutrino detector like KM3NeT/ORCA, the zenith and azimuth angle of a local coordinate system can be used to describe directions. The atmospheric neutrino flux can be approximated very well as being constant along its azimuth coordinate. The more interesting coordinate is the zenith angle that describes if a neutrino comes from above or below the local horizon. If a neutrino enters the detector from directly above, the propagation length L is minimal with a value of about 15 km. If it comes directly from below, the neutrino has to travel from the atmosphere to the ground and then through the whole Earth. When working with atmospheric neutrinos, the propagation length L is mainly substituted by the zenith angle θ or its cosine $\cos\theta$, respectively.

As explained in section 1.3, the oscillation probabilities of the transition $\nu_{\mu} \rightarrow \nu_{\tau}$ depend on the second and third row of the PMNS matrix, making this transition a good candidate to constrain the matrix elements in these two rows further, which is needed for the experimental test of the unitarity of the PMNS matrix. Since practically no tau neutrinos are created in the atmosphere, all tau neutrinos detected in the GeV energy regime must exist due to the transitions $\nu_{\mu} \rightarrow \nu_{\tau}$ and $\nu_{e} \rightarrow \nu_{\tau}$ (analogous for antineutrinos). The oscillation probabilities for transitions into a tau neutrino are

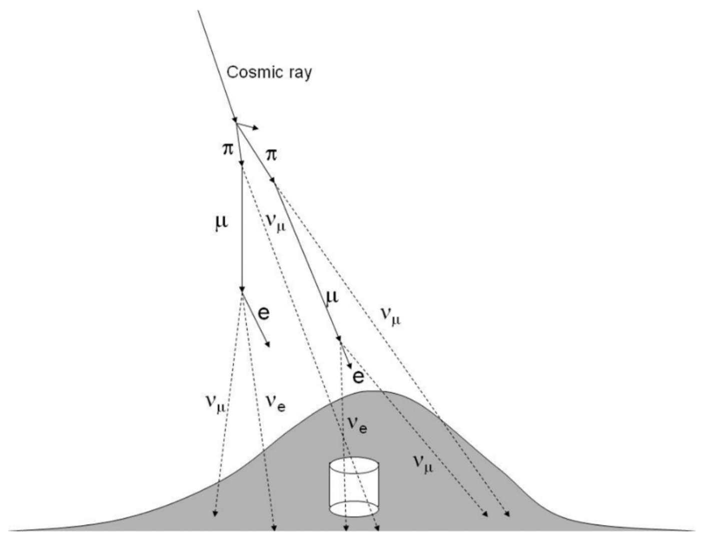


Figure 1.4.: The interaction chain induced by cosmic rays interacting with nuclei in Earth's atmosphere is shown. The white cylinder represents the Super-Kamiokande detector built below a mountain providing shielding from undecayed muons. From [13].

visualized in fig. 1.5. It can be concluded from this plot that most of the tau neutrinos in the atmospheric neutrino flux result from the transitions $\nu_\mu \to \nu_\tau$ and $\bar{\nu}_\mu \to \bar{\nu}_\tau$, which has large regions of high transition probabilities for a variety of energies and zenith angles. In contrast, transitions from electron neutrinos to tau neutrinos a relatively rare compared to the muon neutrino case.

Hence, it can be concluded that atmospheric neutrinos are suitable to measure the oscillation probabilities of the $\nu_\mu \to \nu_\tau$ transition, from which the PMNS matrix elements can be constrained further. The KM3NeT/ORCA detector will be discussed in the next section, which is a neutrino detector highly sensitive to atmospheric neutrinos.

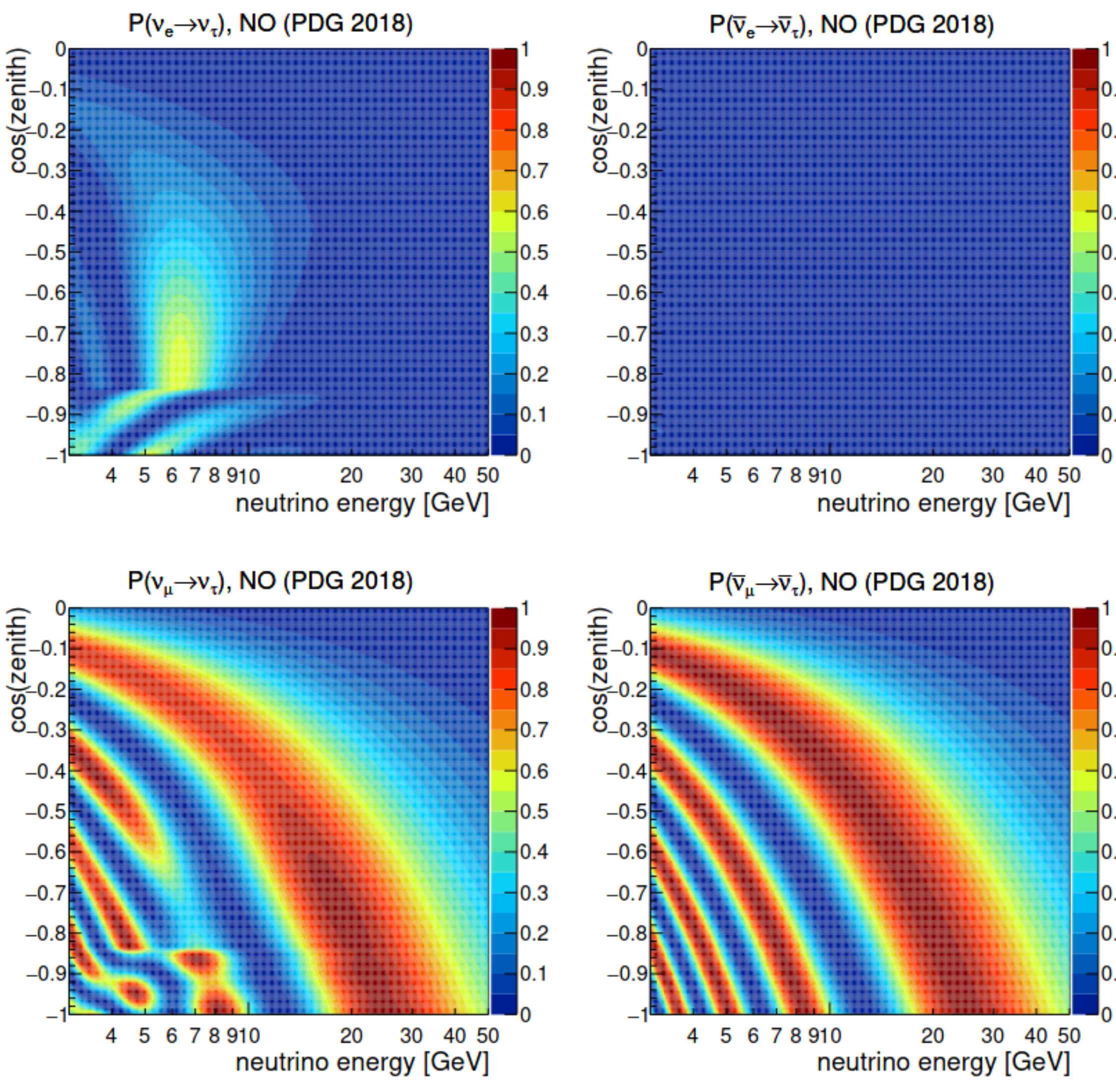


Figure 1.5.: The probabilities for flavor transitions into the tau flavor as a function of the energy and the cosine of the zenith. The calculations assumed a normal ordering (see section 2.1.2). Additional matter effects influencing the oscillation probabilities were taken into account that exceeded the scope of this thesis. For a more detailed explanation and the origin of the plot, see [14].

2. KM3NeT

KM3NeT is an international research collaboration building two next-generation neutrino telescopes in the Mediterranean Sea. The two telescopes are ARCA and ORCA, where ARCA stands for “Astroparticle Research with Cosmics in the Abyss” and ORCA for “Oscillation Research with Cosmics in the Abyss”. This section starts with a short discussion of the research goals of both ARCA and ORCA, but the focus lies more on ORCA since this is the detector optimized for neutrino oscillation physics. After that, the detector design and the detection principle will be explained, followed by a discussion of the event signatures that neutrinos of different flavors induce in the detector.

2.1. Science goals

2.1.1. Astroparticle physics with KM3NeT/ARCA

KM3NeT/ARCA is a detector that will be used to advance the field of neutrino astronomy. As shown in fig. 1.3, active galactic nuclei (AGNs) are suspected to be a source responsible for a small flux of high-energy neutrinos in the TeV-PeV energy regime, which was detected for the first time by IceCube in 2013[15]. These AGNs are, additionally to being a neutrino source, also suspected to be cosmic ray accelerators. Neutrinos are ideal messengers for information about their sources since they can travel large distances through the universe without being deflected by electromagnetic fields in contrast to, e.g., protons. Additionally, they can traverse through high matter densities in astrophysical objects much better than photons. Hence, even a few high-energy neutrinos with a sufficiently well-reconstructed direction would provide valuable information about their origin.

KM3NeT/ARCA focuses on finding the neutrino flux from the cosmic ray accelerators in our galaxy[16]. The strategy for identifying these neutrino events is to identify muons traveling upward in the detector. These muons must have been produced by

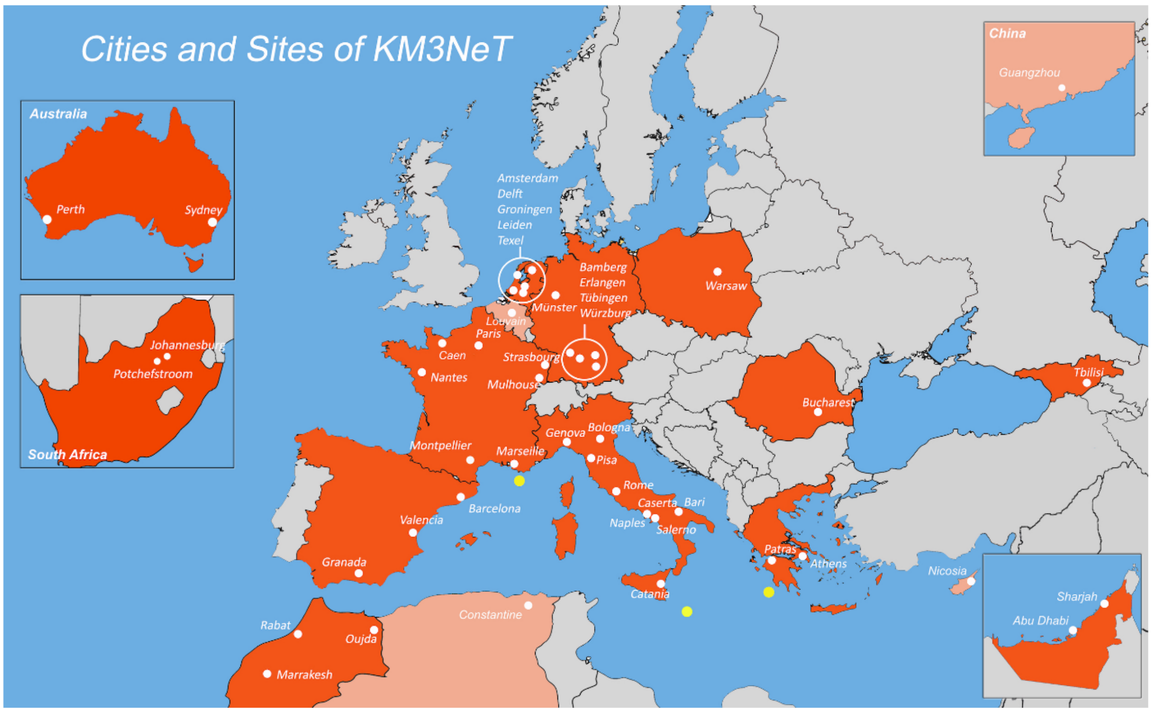


Figure 2.1.: A map showing important locations of KM3NeT. The KM3NeT installation sites are shown in yellow. ORCA is currently built at the French site, while the ARCA detector is built on the Italian site. A third suitable site is available in Greece (courtesy KM3NeT).

neutrino interactions since only neutrinos can traverse the Earth without being absorbed. Additionally, as we will see later in section 2.3, charged muons leave so-called track events in the detector from which the neutrino's direction can be reconstructed best. Since most potential galactic neutrino sources are located in the southern sky, the Mediterranean Sea is ideal for such an experiment. Hence, it was decided to build the ARCA detector about 100 km offshore from Porto Palo di Capo Passero, Sicily. An overview of the KM3NeT installation sites can be seen in fig. 2.1.

2.1.2. Oscillation physics with KM3NeT/ORCA

KM3NeT/ORCA is the detector that is used for measurements interesting in the field of particle physics. ORCA is optimized to detect neutrinos with lower energies than ARCA, namely the atmospheric neutrino flux described in section 1.4. This specialization will enable ORCA to measure the energy- and zenith-dependent oscillation probabilities,

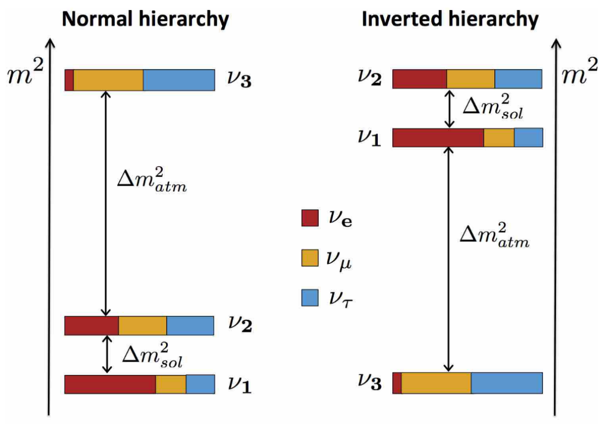


Figure 2.2.: The two possible mass orderings are shown. From [16].

which contributes to clarifying some open questions like, e.g., the unitarity of the PMNS matrix discussed in section 1.3.

Although further constraining the PMNS matrix elements is an important goal for KM3NeT/ORCA, the main goal of the detector is to clarify another still open question, namely the determination of the neutrino mass hierarchy. As shown in eq. (1.11), neutrino oscillation experiments are only sensitive to squared mass differences, not individual masses. In the three flavor theory described in section 1.2, there exist two independent squared mass differences Δm^2_{21} and Δm^2_{31}. The squared mass difference Δm_{21} is constrained today to a value of $7.41^{+0.21}_{-0.20} \cdot 10^{-5}\,\mathrm{eV}^2$ from the data of solar and long-baseline reactor experiments[10, 11]. In contrast, the sign of the squared mass difference Δm^2_{31} is still unknown. The current best-fit value for the case that it is positive is $2.511^{+0.028}_{-0.027} \cdot 10^{-3}\,\mathrm{eV}^2$. For the negative case, the best-fit value is $-2.498^{+0.032}_{-0.025} \cdot 10^{-3}\,\mathrm{eV}^2$. In both cases, the absolute value is about two orders of magnitudes larger than Δm^2_{21}, which leads to two different mass orderings visualized in fig. 2.2. These two cases are called either the "normal hierarchy" for the $m_1 < m_2 < m_3$ case and the "inverted hierarchy" for the $m_3 < m_1 < m_2$ case. The ordering of the neutrino mass hierarchy is significant from a theoretical and experimental point of view for multiple reasons. One of the important reasons is that the constraints on the PMNS matrix elements are affected by whatever hierarchy turns out to be the correct one.

Analogous to KM3NeT/ARCA, the ORCA detector is also built in the Mediterranean Sea. Its installation site is 40 km offshore from the French city Toulon, which can be seen in fig. 2.1. This site is about 10 km west of the former neutrino telescope ANTARES[16], for which KM3NeT can be seen as a successor experiment.

2.2. Detection principle and detector design

Both KM3NeT/ARCA and KM3NeT/ORCA use the same detector hardware, but ORCA is instrumented much denser than ARCA. This dense instrumentation is necessary for ORCA to detect atmospheric neutrinos in the GeV range, which is many orders of magnitudes smaller than the TeV to PeV cosmic neutrinos that ARCA is supposed to detect. Since this thesis aims to identify tau neutrinos in the atmospheric neutrino flux, the relevant detector for this work is ORCA, which will be the focus from now on.

KM3NeT/ORCA is a water-Cherenkov detector located in the Mediterranean deep sea at a depth of about 2450 m[16]. The large amount of water above the detector provides shielding against sunlight and other disturbing particles like atmospheric muons that did not decay on their way to the ground. Both the detection principle and the detector design can be seen in fig. 2.3. In this image, a neutrino that traveled through the Earth interacts close to the detector, which causes the emission of secondary charged particles. These charged particles are typically faster than the speed of light in water, resulting in the emission of Cherenkov light. A more detailed overview of these interactions and the resulting event topologies will be given in section 2.3. ORCA is built to detect this Cherenkov light in a way that makes it possible to reconstruct physical quantities like the energy, the zenith angle, and the flavor of the incident neutrino with sufficient precision during the subsequent data analysis.

As can be seen in fig. 2.3, the ORCA detector consists of an array of so-called “Digital Optical Modules” (DOMs), which are pressure-resistant glass spheres of 3-inch diameter. Each of these DOMs houses 31 photomultiplier tubes (PMTs) oriented in different directions, which can detect the Cherenkov light produced due to a neutrino interaction. The DOMs are grouped in so-called “Detection Units” (DUs), where each DU consists of 18 DOMs. These DOMs are connected by an electro-optical cable transmitting power and data between the DOMs. Each DU is anchored to the seabed and holds its 18 DOMs in a close-to-vertical orientation due to its buoyant top. The ORCA detector consists of one “building block”, which consists of 115 DUs. The average horizontal spacing between two DUs in ORCA is about 20 m, while the average vertical spacing between two DOMs of the same DU is about 9 m[16]. The DUs are connected to so-called junction boxes, which are connected via electro-optical cables transmitting power from and data to a shore station. With this setup, the lower energy threshold for a neutrino detection is about 3 GeV for ORCA.

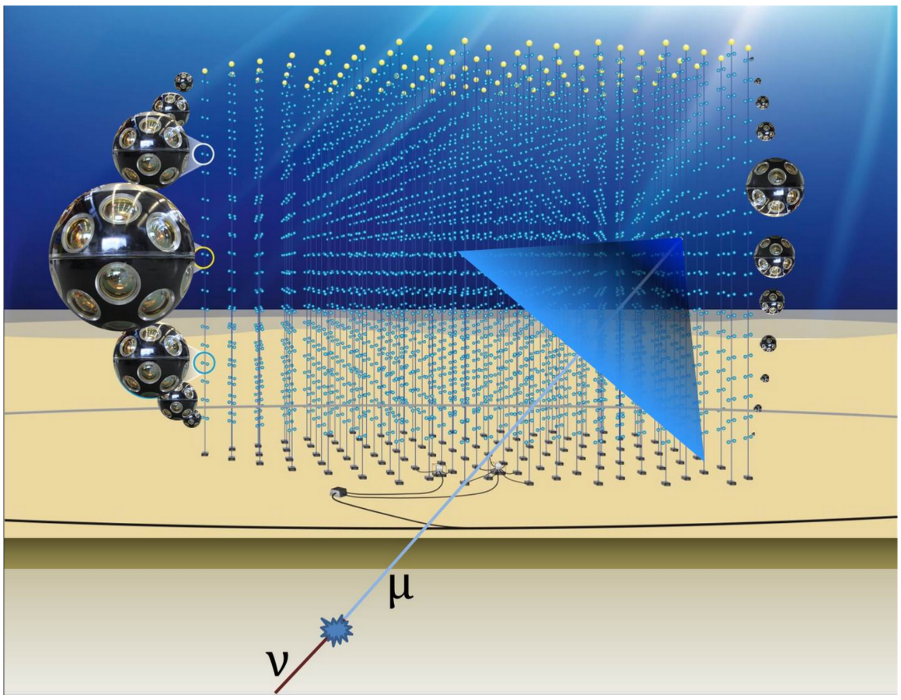


Figure 2.3.: The detection principle and the design of the ORCA detector are shown (courtesy KM3NeT).

The full ORCA detector is still under construction and is expected to be built in the next few years. At the time of this writing, 15 of the 115 planned DUs are already deployed in the water and taking data. When this thesis was started, ORCA consisted only of 6 DUs, which is why Monte Carlo simulations from ORCA6 are used to train the Graph Neural Networks in later sections.

2.3. Neutrino event topologies

As explained in section 1.4, practically all tau neutrinos measured in the atmospheric neutrino flux are the result of neutrino oscillations, for which the $\nu_\mu \to \nu_\tau$ transition is the most dominant (see fig. 1.5). In section 1.3, it was shown that this transition is of high significance because it can further constrain the atmospheric the second and third row of the PMNS matrix, which is necessary to verify the unitarity of the PMNS matrix. Hence, one has to find ways to identify the events in the ORCA detector induced by tau neutrinos so that they can be used for further analyses. This task is known as “tau identification”, and it has achieved the status of being notoriously tricky for reasons explained in this section.

Neutrinos can weakly interact with nuclei by either neutral current (NC) interactions mediated by the neutral Z^0 boson or by charged current (CC) interactions mediated by the charged $W^{\pm}$ bosons. In all of these interactions, the neutrino transmits some of its energy to a quark in the nucleus, which ejects it from the nucleus. Due to the effect of color confinement, new hadronic particles are created, resulting in the emission of a hadronic jet. This jet consists of fast and charged particles, resulting in the radiation of Cherenkov light called a hadronic shower. Since these hadrons have a short interaction length or decay length compared to the size of the ORCA detector, the Cherenkov light is emitted from an effective point-like source[14].

All three flavors can interact via NC interactions ν^{NC}, which are schematically visualized in fig. 2.4 in the top left image. The only light deposited in the detector by this interaction type is the light of the hadronic jet described above since the outgoing neutrino does not carry an electric charge. Since the Cherenkov light comes solely from showers, this type of event is called a shower-like event or, for short, a shower event.

The CC interactions create a charged lepton in addition to the hadronic jet. The flavor of this charged lepton depends on the flavor of the incoming neutrino. An electron neutrino ν_e causes the additional creation of an electron e^- during a ν_e^{CC} interaction. This electron has a radiation length of only 36 cm in water [17], causing a cascade of photons and $e^{\pm}$ particles that, analogous to the hadronic shower, results in Cherenkov light from an effective point-like source. This cascade of photons and $e^{\pm}$ is called an electromagnetic shower. Again, since only showers deposit light in the detector, this interaction type is a shower event. It is visualized in the middle left image in fig. 2.4.

In contrast, a muon neutrino interacting via a CC interaction causes events known as track-like events or, for short, track events. As a result of a ν_μ^{CC} interaction, charged muons are created that can travel far distances in the detector. During their propagation, they constantly radiate a Cherenkov light cone that can be used well to reconstruct the muon's direction. This interaction type is visualized in the bottom left image of fig. 2.4.

The right side of fig. 2.4 deals with the CC interactions caused by tau neutrinos. These ν_τ^{CC} interactions cause the creation of a tau lepton. This tau lepton has a very short lifetime of only 2.9×10^{-13} s, which means that it decays practically instantaneous in the GeV energy regime for a detector of the size of ORCA[14]. The tau lepton has three different decay channels that can be seen on the right side of fig. 2.4. The first

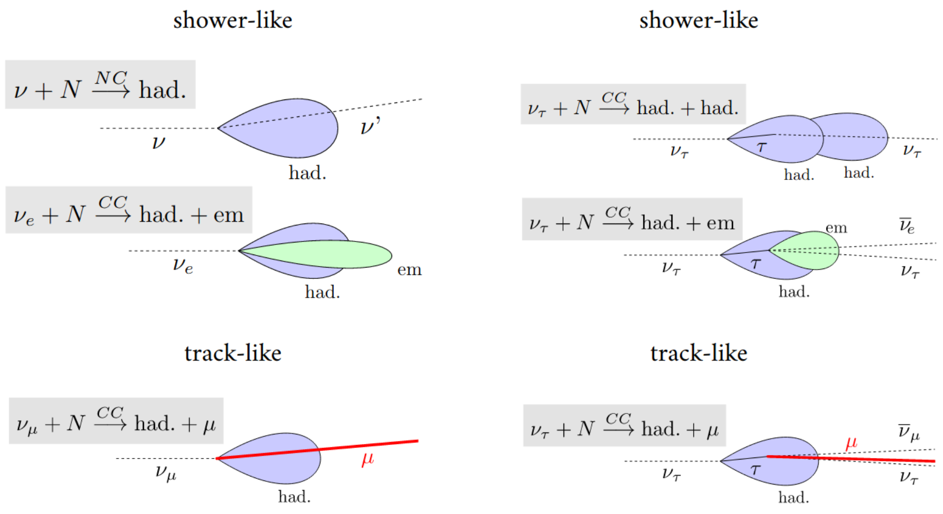

shower-like
shower-like
$\nu + N \xrightarrow{NC}$ had.
ν
ν'
had.
$\nu_\tau + N \xrightarrow{CC}$ had. + had.
ν_τ
τ
had.
had.
ν_τ
$\nu_e + N \xrightarrow{CC}$ had. + em
ν_e
had.
em
$\nu_\tau + N \xrightarrow{CC}$ had. + em
ν_τ
τ
em
$\overline{\nu}_e$
ν_τ
had.
track-like
track-like
$\nu_\mu + N \xrightarrow{CC}$ had. + μ
ν_μ
μ
had.
$\nu_\tau + N \xrightarrow{CC}$ had. + μ
ν_τ
τ
μ
$\overline{\nu}_\mu$
ν_τ
had.

Figure 2.4.: Comparison of the neutrino event topologies for nontau events on the left and tau events on the right. [14]

decay channel is a decay into hadrons, which causes an additional hadronic shower. This channel is the most probable, with a probability of $\approx 65\%$. The second decay mode is via the creation of an electron, causing the creation of an electromagnetic shower in addition to the hadronic shower. Finally, there exists also a decay into a muon, which causes the event to have a track-like signature in contrast to the other two decay modes leading to shower events. The latter two decays into leptons each have a probability of $\approx 17\%$[14].

At the beginning of this section, it was explained that tau neutrino identification consists of separating events caused by tau neutrinos from events caused by other flavors. This definition can now be refined further. Since all neutrino flavors, including tau neutrinos, can cause ν^{NC} events not distinguishable from each other, the task of tau neutrino identification in this thesis is further specified to the task of identifying events caused by ν_τ^{CC} interactions. From now on, these events are, for simplicity, called "tau events". All the other event types are called "nontau events". As one can see from a direct comparison of the event types on the left side of fig. 2.4 with the events directly to the right, the event topologies of tau events look very similar to the topologies of nontau events, making the task of tau identification very challenging. This problem is tackled in this thesis by the usage of machine learning techniques, in particular, Graph Neural Networks researched in the subfield of Deep Learning, which yielded competitive and often improving performance in related reconstruction tasks using the data of KM3NeT/ORCA (see [18, 19]).

3. Deep learning

Deep Learning (DL) is a subfield of machine learning (ML) that deals with extracting information from raw data using so-called neural networks. Although many of the foundations and neural network architectures used today were already known since the late 20th century, the lack of computing hardware with sufficient performance did not make the approach feasible for usage in other fields of academia and industry until the early 2010s. Since then, neural networks have begun to outperform other ML techniques in various tasks, making them more interesting and accessible to other fields. In the context of physics, they are now widely applied to extract information from the raw data collected by detectors, helping experimentalists reconstruct physical quantities and increase the sensitivity of detectors to physical effects. This section starts by explaining what a neural network is, how it is structured, and what the general idea behind them is. After that, different building blocks of neural networks will be introduced step by step alongside an explanation of standard training, evaluation, and regularization procedures. In the end, so-called Graph Neural Networks will be discussed, which will be applied to tau neutrino identification in KM3NeT/ORCA during the rest of this thesis.

3.1. Structure of neural networks

A good start for understanding neural networks is to look at the simplest type of neural network and analyze its components step by step. The simplest type of neural network is a so-called multilayer perceptron (MLP). A diagram showing the structure of a typical MLP can be seen in fig. 3.1. MLPs consist of three kinds of layers: an input layer, several hidden layers, and an output layer stacked on top of each other. Each layer consists of a set of so-called nodes, with each node representing one real number. Each node of a layer is connected to each node of the previous layer, which is why this type of layer is often called "dense layer". These connections define a flow of information through the

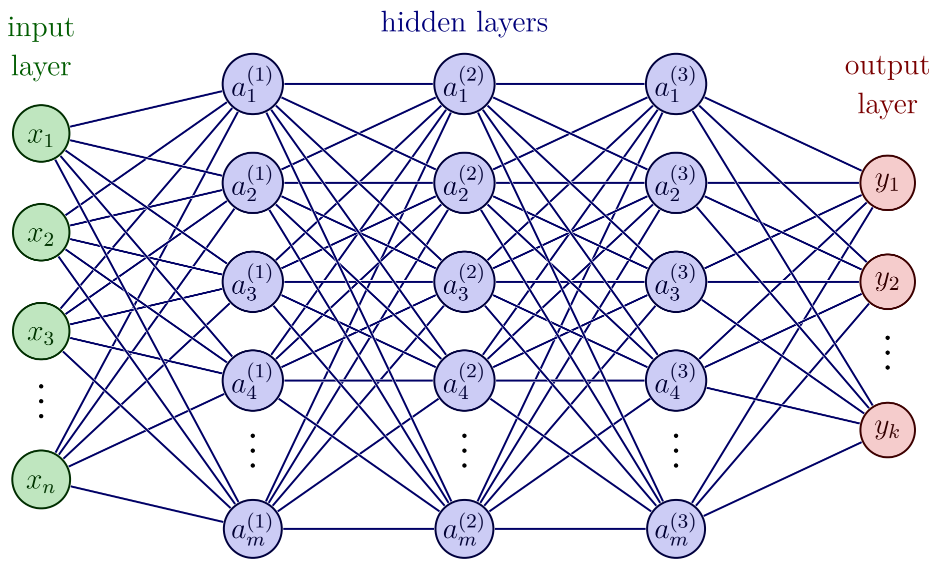


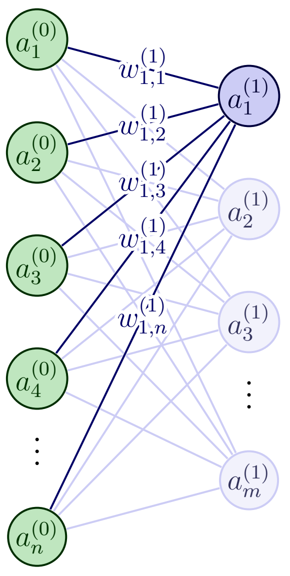
Figure 3.1.: Diagram of a multilayer perceptron, the simplest type of neural network. Adapted from [20].

neural network that will be defined more precisely below.

The "input layer" consists of a set of nodes, i.e., a set of real numbers that has to be provided to the neural network. This input represents the raw data from which information should be extracted. In principle, each object representable by a tuple of real numbers can be supplied as input to a neural network. This includes digitized information like images, speech recordings, and text. In this thesis, the input data consists of neutrino events measured with the KM3NeT/ORCA detector, which can be represented by graphs. A detailed explanation of the data representation of these neutrino events will be given in section 3.5.1.

The "output layer" consists of a set of nodes representing the neural network's desired output. Again, this could be any object representable by a tuple of real numbers. In this thesis, the output consists of a so-called "tau score", which can be represented by a single neuron. The tau score is a real number between zero and one, indicating how sure the network is that the supplied event is a tau event. Hence, if the neural network were a perfect classifier, it would assign every tau event a tau score of one and every nontau event a score of zero.

Every neural network in the field of Deep Learning has at least one "hidden layer". The central idea behind Deep Learning is that neural networks should be able to build an



$$= \sigma\left(w_{1,1}^{(1)} a_1^{(0)} + w_{1,2}^{(1)} a_2^{(0)} + \ldots + w_{1,n}^{(1)} a_n^{(0)} + b_1^{(1)}\right)$$

$$= \sigma\left(\sum_{i=1}^{n} w_{1,i}^{(1)} a_i^{(0)} + b_1^{(1)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & \ldots & w_{1,n}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & \ldots & w_{2,n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(1)} & w_{m,2}^{(1)} & \ldots & w_{m,n}^{(1)} \end{pmatrix} \begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_m^{(1)} \end{pmatrix}\right]$$

$$a^{(1)} = \sigma\left(\boldsymbol{W}^{(1)} a^{(0)} + \boldsymbol{b}^{(1)}\right)$$

Figure 3.2.: This figure illustrates the operation used by a multilayer perceptron to calculate the nodes in the kth from the nodes in the previous layer. In this example, the nodes of the first hidden layer are calculated from the input nodes. Adapted from [20].

internal hierarchy of concepts that enables them to solve their task. The network should find simple concepts to structure the raw input data, which can then be combined into more complex and abstract concepts. The name "Deep Learning" comes from the idea that the learned hierarchy of concepts, when laid out as a graph showing how the complex concepts are formed out of the simpler concepts, should be deep [21]. This idea of a deep hierarchy of concepts is imprinted in every deep neural network by giving them at least one hidden layer. Each layer in the network can then use the information from the previous layer to form a more suitable representation of the data, which is then used by the next layer to do the same, and so on until one arrives at the output layer.

The mathematical operation used by MLPs to calculate the nodes in its kth layer, denoted as $a^{(k)}$, from the previous layer $a^{(k-1)}$ is illustrated in fig. 3.2. The ith node in the kth layer, denoted by $a_i^{(k)}$, is connected by an edge to each node $a_j^{(k-1)}$ in the previous layer. Each edge represents a weight, which is again a real number. The weight of the edge that connects the nodes $a_i^{(k)}$ and $a_j^{(k-1)}$ is denoted as $w_{i,j}^{(k)}$. Additionally, each node in the kth layer has a so-called bias, which is also a real number. The bias of the ith node in the

kth layer will be denoted as $b_i^{(k)}$. The operation that calculates node $a_i^{(k)}$ is defined as

$$a_i^{(k)} = \sigma\left(\sum_{j=1}^{n} w_{i,j}^{(k)} a_j^{(k-1)} + b_i^{(k)}\right), \tag{3.1}$$

where n is the number of nodes in the $(k-1)$th layer, and σ is the so called "activation function". Since the term in the argument of σ in eq. (3.1) is linear in the values of the previous layers' nodes $a_j^{(k-1)}$, a nonlinear activation function is needed such that the kth is not limited to be a linear transformation of the $(k-1)$th layer. The most popular choice for the activation function is the "rectified linear unit", in short ReLU, which is defined as

$$\sigma_{\text{ReLU}}(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

This activation function will be used during this thesis for each node except for the output node, for which the "sigmoid" activation function will be used, defined as

$$\sigma_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}}. \tag{3.3}$$

Since the sigmoid function is bounded to values between zero and one, it is ensured that the tau score can be interpreted as a probability-like quantity. As shown in fig. 3.2, eq. (3.1) can be compactly written for a parallel calculation of the whole kth layer by the use of matrix formalism

$$\begin{pmatrix} a_1^{(k)} \\ a_2^{(k)} \\ \vdots \\ a_m^{(k)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,1}^{(k)} & w_{1,2}^{(k)} & \dots & w_{1,n}^{(k)} \\ w_{2,1}^{(k)} & w_{2,2}^{(k)} & \dots & w_{2,n}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1}^{(k)} & w_{m,2}^{(k)} & \dots & w_{m,n}^{(k)} \end{pmatrix} \begin{pmatrix} a_1^{(k-1)} \\ a_2^{(k-1)} \\ \vdots \\ a_n^{(k-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_m^{(k)} \end{pmatrix}\right], \tag{3.4}$$

where m is the number of nodes in the kth layer. This can be written even more compactly as

$$a^{(k)} = \sigma\left(\boldsymbol{W}^{(k)} a^{(k-1)} + \boldsymbol{b}^{(k)}\right). \tag{3.5}$$

The nodes of the kth layer are therefore calculated by the nonlinear function $f^{(k)} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, $x \longmapsto f_{\Theta_k}^{(k)}(x)$ defined in eq. (3.5), where Θ_k denotes the set consisting of all weights $\boldsymbol{W}^{(k)}$ and biases $\boldsymbol{b}^{(k)}$. Stacking layers on each other is equivalent to chaining these functions together. Hence, a neural network consisting of k layers, where the input layer is the zeroth layer with n input nodes, and the output layer is the $(k-1)$th layer with m output nodes, can be understood as a function

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m,\; x \longmapsto f_\Theta(x) = f_{\Theta_{k-1}}^{(k-1)}\left(f_{\Theta_{k-2}}^{(k-2)}\left(\dots\left(f_{\Theta_0}^{(0)}(x)\right)\right)\right), \tag{3.6}$$

where $\Theta = \bigcup_{i=0}^{k-1} \Theta_i$ is the set of all weights and biases in the network. For simplicity, Θ will, from now on, just be called the set of all weights, and the biases are implicitly included in that terminology.

The goal of Deep Learning is to find suitable weights Θ such that the function f_Θ defined in eq. (3.6) produces a desired output for a given input sample. This goal can be achieved by providing a dataset to the neural network consisting of many input samples with the corresponding desired output. The desired output of an input sample is commonly called the “label” of that sample. The label must be provided by some extern entity, e.g., a human expert. In this context, the network takes on the role of a student trying to understand the task from the data provided by this outside entity, which could therefore be seen as a kind of teacher. For this reason, this approach of training a neural network is commonly called “supervised learning”. The following section will discuss the building blocks and the algorithms used for this weight optimization.

3.2. Training

Loss functions

Machine learning models $f_\Theta(x)$ optimize their weights Θ by an iterative process called “training”. For this process, one has to provide a training dataset $\boldsymbol{X}^{\text{train}} = \{(x_i, y_i) \,|\, i = 1, \ldots, N\}$ consisting of training samples x_i and corresponding labels y_i. The model will produce a prediction $y_i^{\text{pred}} = f_\Theta(x_i)$ for each of the training samples x_i, which is then compared to the label y_i to get information on how to modify the weights Θ to produce an output closer to y_i. A so-called “loss function” is used to compare all predictions y_i^{pred} to the true labels y_i.

A loss function $L : \mathbb{R}^{|\Theta|} \longrightarrow \mathbb{R}^+$, $\Theta \longmapsto L(\Theta)$ is a function that maps the vector containing all weights Θ of the model to a positive real number, which can be interpreted as the mean distance between the vector of true labels $\boldsymbol{y}$ and the model's predictions $\boldsymbol{y}^{\text{pred}}(\Theta)$ over the whole training set $\boldsymbol{X}^{\text{train}}$. Hence, the loss function over the training dataset $\boldsymbol{X}^{\text{train}}$ can be defined as

$$L(\Theta) = \frac{1}{N} \sum_{(x,y) \in \boldsymbol{X}^{\text{train}}} l(y, f_\Theta(x)), \tag{3.7}$$

where $l(y, f_\Theta(x))$ is measuring the distance between the label y and the prediction $y^{\text{pred}} = f_\Theta(x)$. Since this thesis studies a binary classification problem, i.e., the problem has two mutually exclusive labels “tau” and “nontau”, a popular choice for the loss function is the “binary cross-entropy” (BCE), which is defined as

$$L_{\text{BCE}}(\Theta) = -\frac{1}{N} \sum_{(x,y)\in \boldsymbol{X}^{\text{train}}} y \log f_\Theta(x) + (1-y)\log(1 - f_\Theta(x)). \tag{3.8}$$

In this loss function, each tau event adds a value of $-\log y^{\text{pred}}$ to the loss function since $y = 1$ for tau events. Hence, a perfect prediction $y^{\text{pred}} = 1$ would lead to no increase of the loss function since $\log(1) = 0$. On the other hand, since the logarithm approaches $-\infty$ for values closer to zero, a bad prediction $y^{\text{pred}} << 1$ would result in a large, positive term $-\log y^{\text{pred}}$ added to the loss function. The argumentation is analogous for a nontau event with a label $y = 0$.

Gradient descent and learning rate

Gradient descent is an iterative method to find a local minimum of a differentiable function. This method can be used to find suitable weights that minimize the loss function, i.e., the mean distance between the model’s predictions and the labels over the training set $\boldsymbol{X}^{\text{train}}$.

The loss function is differentiable with respect to the weights Θ since $f_\Theta(x)$ is a composition of functions that are themselves differentiable in all weights $w \in \Theta$ as defined in eq. (3.6). The idea of gradient descent is to make small iterative steps in the opposite direction of the gradient of $L(\Theta)$, which is taken with respect to each weight $w \in \Theta$. This procedure works because the direction opposite to the gradient is the direction in which the function has its steepest decrease. Hence, the weights Θ_t after the tth iterative step in gradient descent can be written as

$$\Theta_t = \Theta_{t-1} - \eta \nabla_\Theta L(\Theta_{t-1}), \tag{3.9}$$

where ∇_Θ is the gradient with respect to the weights and η is the so-called “learning rate”. The learning rate is a parameter that has to be chosen before the training starts and is not optimized during training. Parameters with these properties are called “hyperparameters”. Small learning rates ensure that the trajectory of the steepest descent is followed closely, but the algorithm needs more iterations until a good convergence to a local minimum is reached. On the other hand, too large learning rates can cause the algorithm to move past a local minimum and prevent the algorithm from converging to a local minimum. Each step in gradient descent requires a calculation of the gradient

$$\nabla_\Theta L\left(\Theta\right) = \frac{1}{N} \sum_{(x,y)\in \boldsymbol{X}^{\text{train}}} \nabla_\Theta l(y, f_\Theta(x)), \tag{3.10}$$

which in turn requires a calculation of the gradient $\nabla_\Theta l(y, f_\Theta(x))$ for all training samples x. Calculating the gradient over all samples in the training dataset is computationally very expensive, which is why “stochastic gradient descent” (SGD) is used as an approximation for gradient descent.

In SGD, the gradient is not calculated over the entire training dataset $\boldsymbol{X}^{\text{train}}$, but over smaller subsets of it which are called “batches”. Each batch has the same “batch size” b that has to be chosen before the training starts. Hence, the batch size is a hyperparameter. The batches are sampled from $\boldsymbol{X}^{\text{train}}$ without replacement. The tth iteration approximates the loss function using the tth batch denoted as $\boldsymbol{X}_t^{\text{train}}$, i.e.,

$$L_t\left(\Theta\right) = \frac{1}{N} \sum_{(x,y)\in \boldsymbol{X}_t^{\text{train}}} l(y, f_\Theta(x)). \tag{3.11}$$

The gradient $\nabla_\Theta L_t$ for the tth iteration is calculated on this approximation of the loss function and then used for the weight update in eq. (3.9), which leads to the new weight update rule for SGD

$$\Theta_t = \Theta_{t-1} - \eta \nabla_\Theta L_t\left(\Theta_{t-1}\right)., \tag{3.12}$$

This operation is not only computationally cheaper, but it also has the advantage that the noise added due to the approximations could “kick” the algorithm out of local minima and let it converge to a deeper, more global minimum.

Adam optimizer

The weight update rule in eq. (3.12) has a relatively simple structure that is not optimal in many situations. For this reason, many so-called “optimizers” were developed that improve the weight update rule of SGD. This thesis uses the “Adam” optimizer as introduced in [22].

Adam is derived from “adaptive moment estimation”. It uses exponential moving averages of the first and second moments of the gradients in its update rule. As defined in [22], in each iteration t, the following operations are performed:

$$g_t \leftarrow \nabla_\Theta L_t(\Theta_{t-1}) \tag{3.13}$$

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{3.14}$$

$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2. \tag{3.15}$$

The first operation calculates the gradient of the loss function on batch t evaluated at the weights calculated in the previous step Θ_{t-1}. The second step is an exponential moving average over the gradients, which is a biased estimator for the mean. Similarly, the third operation is an exponential moving average over the squared gradients, a biased estimator of the uncentered variance. The second (third) operation uses the hyperparameter β_1 (β_2) to control the exponential decay rate, and the recommended value for this hyperparameter is 0.9 (0.999). Finally, the following corrections are applied to remove biases from the exponential moving averages:

$$\widehat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t} \tag{3.16}$$

$$\widehat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}. \tag{3.17}$$

This results in the update rule of the Adam optimizer

$$\Theta_t \leftarrow \Theta_{t-1} - \eta \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}, \tag{3.18}$$

where ϵ is a parameter included to provide numerical stability if $\sqrt{\widehat{v}_t}$ is close to zero. The value suggested by the authors is $\epsilon = 10^{-8}$. The weight update is proportional to the exponential moving average of the gradients $\widehat{m}_t$, which means that the stepsize of an update is larger if previous updates were large and smaller if previous updates were small. The square root of the exponential moving average estimating the uncentered variance $\sqrt{\widehat{v}_t}$ is in the denominator, meaning that the stepsize of a weight update should be smaller if previous gradients scattered widely around their mean. On the other hand, if the previous gradient updates point in roughly the same direction, the stepsize is allowed to be larger.

The following sections use Adam with the default value for β_1 and β_2, but the value of ϵ is set to 0.1 as recommended, e.g., in the documentation of TensorFlow[23].

3.3. Metrics

The loss function is used during training to measure the deviation of the model's predictions to the true labels in a differentiable way with respect to the model's weights. The latter point is essential for being able to calculate the gradient of the loss function with respect to the weights, which is crucial for the application of SGD. However, one often wants to evaluate the model's performance with quantities that are not differentiable with respect to the model's weights. These quantities are called "metrics" in Deep Learning.

Accuracy

A simple example of a metric in a binary classification problem is the "accuracy". The accuracy is calculated by dividing the number of correctly classified samples by the total number of samples. In binary classification problems, one often introduces the four categories of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). In this terminology, the terms "positive" and "negative" refer to whether a sample was labeled positive or negative by the classifier. When using a neural network with one output neuron as a classifier, one sets a threshold t above which a prediction is defined as positive. For example, if one uses a neural network using neutrino events as input and giving a tau score as output as defined in section 3.1, one could define the threshold as $t = 0.5$. If the network prediction for a particular sample is larger than 0.5, the prediction is interpreted as a tau event, otherwise as a nontau event. The terms "true" and "false" specify if the network's prediction matches or contradicts the sample's true label. Hence, using this terminology, the accuracy can be defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}. \tag{3.19}$$

Precision and recall

"Precision" and "recall" are metrics widely used in binary classification problems. Precision is often called "purity" in the physics literature. When using the above terminology, it is defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3.20}$$

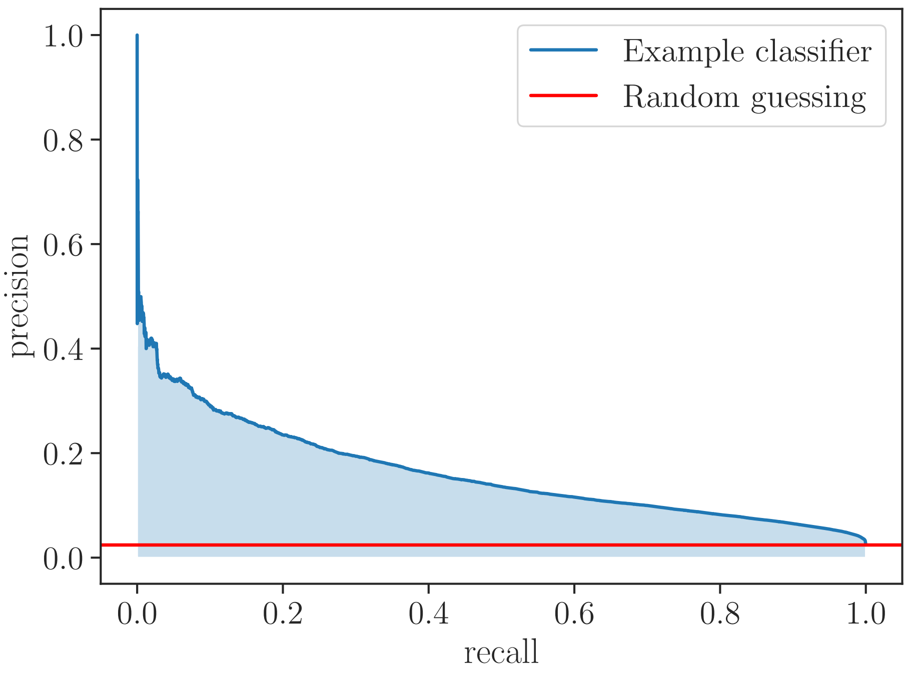


Figure 3.3.: An example of a precision-recall curve. The blue curve corresponds to an arbitrary example classifier, while the red curve corresponds to a randomly guessing classifier.

Hence, it is the probability that a positively predicted sample in the dataset is indeed a true positive. In tau identification, precision can be understood as the probability that a positively labeled neutrino event is a tau event.

On the other hand, the recall, also often called the “efficiency”, is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.21}$$

Hence, it can be interpreted as the probability that a sample labeled as positive will be predicted as positive. In tau identification, recall is the probability that a tau event will be recognized as a tau event by the classifier.

PR-AUC

Both precision and recall are dependent on the particular threshold that one uses. One can visualize this dependency by drawing a “precision-recall curve”, in which each point of the curve corresponds to one particular threshold $t \in [0, 1]$. An example of a precision-recall curve is shown in fig. 3.3, where the tradeoff between precision and recall can be seen. A perfect classifier would classify each tau event as a one and each nontau event as a zero, which would result in a precision-recall curve that consists solely of the point $(\text{precision}, \text{recall}) = (1, 1)$. A randomly guessing classifier, on the other hand, would predict a random number between zero and one for each sample. Hence, in the set consisting of all positively predicted samples, the fraction of tau events would be the same as in the whole dataset. From this, it follows that the precision remains constant for all thresholds in the case of a randomly guessing classifier, and its value corresponds to the fraction of tau events in the whole dataset. The red curve in fig. 3.3 represents a randomly guessing classifier.

The choice of the classifier's threshold depends on the concrete physics analysis that should be performed with it. In some analyses, higher precision is desirable at the expense of a lower recall. In other analyses, the number of samples belonging to the positive class is so small that a higher recall is necessary. Hence, since it is not yet known which threshold will be preferred by subsequent analyses using the classifiers trained in this thesis, a threshold-independent metric will be used to evaluate the networks' performance. A popular choice for a threshold-independent metric is the "precision-recall area-under-the-curve" (PR-AUC). The PR-AUC is shown in fig. 3.3 as the blue area under the precision-recall curve. The PR-AUC is fixed in the range $[0, 1]$, where the PR-AUC $= 0$ case happens if a model predicts every sample to be zero, resulting in a precision-recall curve that consists solely of the point $(0, 0)$. In contrast, as discussed above, the PR-AUC $= 1$ case corresponds to a perfect classifier.

3.4. Regularization

The goal of machine learning is to train a model that can make good predictions on data that it has never seen before, in contrast to a model that is only good on the data used for training. This ability to generalize is measured during the training process by a periodic evaluation on a "validation set" $\boldsymbol{X}^{\text{val}}$ containing data that is not in the training set $\boldsymbol{X}^{\text{train}}$. This measurement happens typically at the end of one "epoch", defined as one complete run through all the training data sampled in batches without replacement. Typically, after improvements on both the training and validation data during the first couple of epochs, the model's performance on the validation set starts to deteriorate after training for further epochs. This effect is known as "overfitting", where the model learns features of the training set by heart and is, therefore, unable to generalize to new data.

The problem of overfitting is commonly tackled by providing either more training data to the model, by reducing the number of free parameters in the model, or by using regularization techniques, which are changes in the machine learning algorithm explicitly aiming to reduce overfitting. This section discusses two regularization techniques, namely "batch normalization" and "dropout".

Batch normalization

"Batch normalization" is a regularization technique that was introduced to tackle the so-called "internal covariate shift"[24]. This shift is defined as the change in a layer's output distributions due to the changes in its parameters during training. Subsequent layers using this output as their input must continuously adapt to the changing input distributions, slowing down the training process. Hence, batch normalization tries to reduce the internal covariate shift by whitening each batch of input data, i.e., setting the mean of each feature over the whole batch to zero and the variance to one. Hence, for a batch $B = \{x_1, \ldots, x_m\}$ consisting of m feature vectors x_i with $x_i^{(k)}$ denoting the kth feature of sample x_i, the following operations are performed:

$$\mu_B^{(k)} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i^{(k)} \tag{3.22}$$

$$\left(\sigma_B^{(k)}\right)^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} \left(x_i^{(k)} - \mu_B^{(k)}\right)^2 \tag{3.23}$$

$$\widehat{x}_i^{(k)} \leftarrow \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \epsilon}}, \tag{3.24}$$

where $\mu_B^{(k)}$ and $\left(\sigma_B^{(k)}\right)^2$ are the sample mean and variance of the batch B. The parameter ϵ is a small number added for numerical stability. As explained in [24], the normalizing operation could constrain the representation power of its inputs. To restore this representation power, two learnable parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are introduced. The output $y_i^{(k)}$ of the batch normalization is finally given by

$$y_i^{(k)} = \gamma^{(k)} \widehat{x}_i^{(k)} + \beta^{(k)}. \tag{3.25}$$

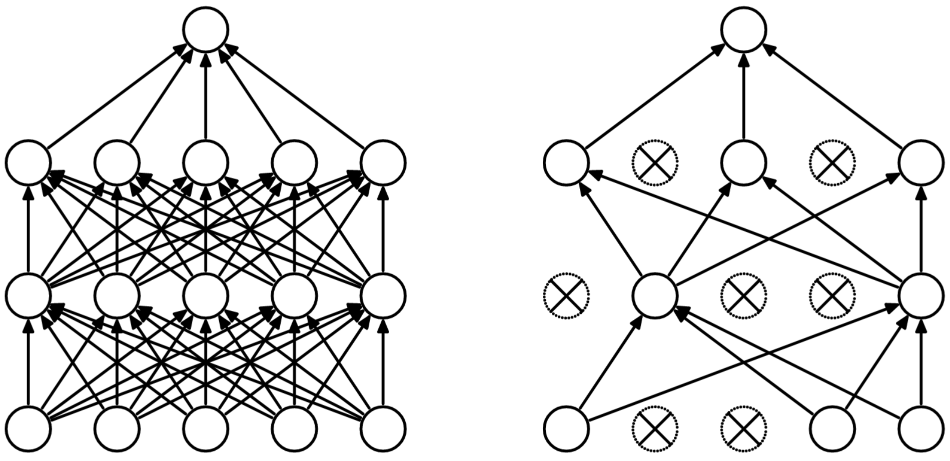


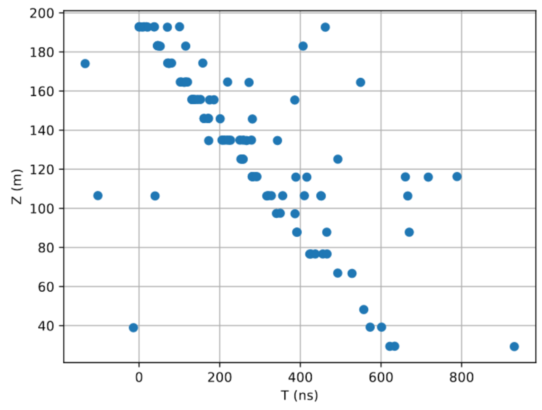
Figure 3.4.: Subfigure (a) shows a MLP. Subfigure (b) shows the thinned MLP that results from the application of dropout, where a fraction p of the neurons was temporarily removed from the training. From [25].

Dropout

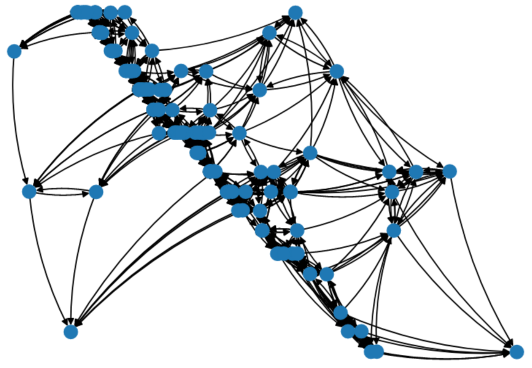
"Dropout" is a regularization technique introduced in [25]. Applying dropout to a layer means temporarily removing a randomly chosen fraction of p neurons in the layer from the training step, thinning out the network in the process. This process is visualized in fig. 3.4. The right subfigure shows a thinned network that can be considered a subnetwork of the whole network.

Applying dropout to a layer is a popular choice for reducing the risk of overfitting in a network. The reasoning behind dropout is that because of the removal of randomly chosen nodes, the nodes that survived the dropout are forced to produce a good result without relying too much on each other. Hence, no complex co-adaptions between a large number of nodes are possible. Instead, the nodes are forced to fulfill the same task by cooperating only with a few other nodes, which is assumed to be more stable on new, unseen data than the complex co-adaptions.

All nodes are used during the validation and inference on new data. A factor of p decreases the layers' weights during validation and inference, which ensures that the expected output of a neuron is the same as during training, where it was removed with a probability of p.



(a) z-t-plot of the event.



(b) Graph representation of the same event.

Figure 3.5.: Subfigure (a) shows the distribution of hits of an event by plotting the z-coordinate over the time-coordinate. Subfigure (b) shows the same event as a graph, where each hit is represented as a node connected to its eight nearest neighbors. From [19].

3.5. Graph Neural Networks

3.5.1. Representing KM3NeT/ORCA data as graphs

As explained in section 2.2, KM3NeT/ORCA consists of a 3D array of DOMs. Each of these DOMs houses 31 PMTs. These PMTs can detect light radiated by secondary particles emitted from a neutrino interaction. An event in KM3NeT/ORCA consists of a set of so-called hits, where each hit corresponds to a measured light signal in one of the PMTs. A hit can be described by the position, time, and the direction in which the PMT recording the hit is facing. The time-over-threshold (ToT) is also stored for each hit, but this quantity is usually omitted. The reason is that the intensity of the light is already encoded in the density of hits measured at close-by PMTs[18]. The direction is usually encoded as a three-dimensional vector with unit length $(\hat{x}, \hat{y}, \hat{z})$. Hence, together with the 3d position (x, y, z) and the time t of the hit, each hit is described by seven real numbers $(x, y, z, \hat{x}, \hat{y}, \hat{z}, t)$.

An event consists of a set of hits distributed in time and space. An example distribution of an event can be seen in fig. 3.5, where the z-coordinate of each hit is plotted over the time-coordinate. Due to their sparse and non-regular distribution, graphs are a suitable representation of this data. Graphs consist of a set of nodes together with a set of edges connecting the nodes. Each hit in an event can be identified with a node in a graph. Therefore, each node has seven node features $\{x, y, z, \hat{x}, \hat{y}, \hat{z}, t\}$. Each node is connected to its k nearest neighbors, where k is a hyperparameter that must be chosen before the graphs can be used to train a neural network. The graph representation is visualized in the right half of fig. 3.5, where the nodes are at the same spots as in the left figure but connected to their eight nearest neighbors. In order to be able to calculate the k nearest neighbors, a distance measure between two nodes has to be defined. This thesis uses, in accordance to [18, 19], the Euclidean distance $\mathrm{d}s^2 = c^2\mathrm{d}t^2 + \mathrm{d}x^2 + \mathrm{d}y^2 + \mathrm{d}z^2$ as a distance measure.

3.5.2. EdgeConv block

This thesis studies the application of "Graph Neural Networks" (GNNs) to tau neutrino identification. A GNN is a neural network that takes graphs as input. The layer that is used in this thesis to operate on the graphs is the "EdgeConv block" as proposed by [26] and implemented for the physics task of jet tagging in [27]. The EdgeConv block performs, similar to a convolutional neural network on regularly spaced images, a convolution over the nodes in a graph connected by an edge.

The EdgeConv block is a layer that takes a graph with F features per node as input and returns a graph with the same number of nodes but with a different number of node features F'. The structure of the EdgeConv block is shown in fig. 3.6. The features $\{x, y, z, \hat{x}, \hat{y}, \hat{z}, t\}$ of each node together with the coordinates $\{x, y, z, t\}$ are supplied to the EdgeConv block, which will use the coordinates to calculate edges as defined in section 3.5.1. Then, the edge features of the edges in the graph are calculated. As explained in [19], for each node $\boldsymbol{n}_i \in \mathbb{R}^{\mathrm{F}}$ connected to the neighbor node $\boldsymbol{n}_j \in \mathbb{R}^{\mathrm{F}}$ by the edge $\boldsymbol{e}_{i,j} \in \mathbb{R}^{2\mathrm{F}}$, the edge features are defined as

$$\boldsymbol{e}_{i,j} = (\boldsymbol{n}_i, \boldsymbol{n}_j - \boldsymbol{n}_i)\,. \tag{3.26}$$

As shown in fig. 3.6, each edge feature vector $\boldsymbol{e}_{i,j}$ is fed to a multilayer perceptron (MLP) as defined in section 3.1, which will be called the "kernel network" from now on. The weights of this kernel network are shared among all nodes in the graph. After each layer of the kernel network, a batch normalization and a ReLU activation function are applied. The kernel network returns an update vector $\boldsymbol{u}_{i,j} \in \mathbb{R}^{\mathrm{F}'}$ as output. The number of neurons in the last layer of the kernel network determines the number of output features

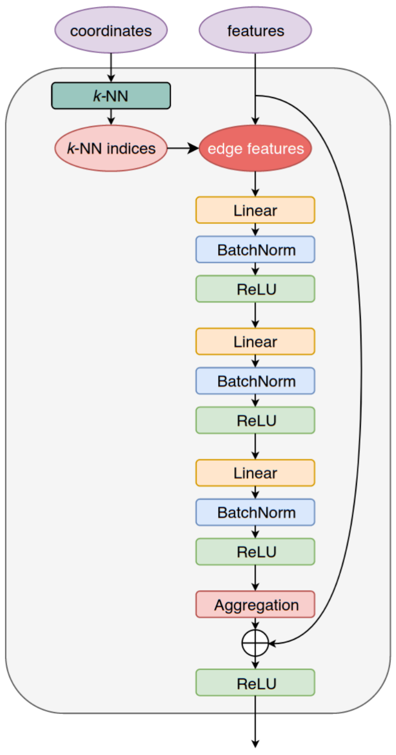


Figure 3.6.: The structure of the EdgeConv block. From [27].

F′. The updated node features $\boldsymbol{n}_i^* \in \mathbb{R}^{\mathrm{F}'}$ are calculated by taking a mean over the update vectors, which gives

$$\boldsymbol{n}_i^* = \frac{1}{k}\sum_{j=1}^{k}\boldsymbol{u}_{i,j}. \tag{3.27}$$

As described in [19], each EdgeConv block can additionally have a shortcut connection passing the original node features $\boldsymbol{n}_i$ through the EdgeConv block without applying the kernel network to them. This operation is implemented by passing the node features $\boldsymbol{n}_i$ through a single dense layer with subsequent batch normalization. The dense layer has the same number of nodes F′ as the updated feature vector $\boldsymbol{n}_i^*$, such that its output can be added elementwise to $\boldsymbol{n}_i^*$.

It should be noted that the aggregation step in eq. (3.27) is invariant under a permutation of the neighboring nodes $\boldsymbol{n}_j$. This symmetry must be built into the network because a graph has no explicit order of its nodes, and two graphs that only differ by order of their nodes should yield the same result after being fed into the GNN.

This thesis uses a slightly modified version of the OrcaNet framework in version v1.0.2 [28]. The modified version will be archived and is planned to be merged into the main branch

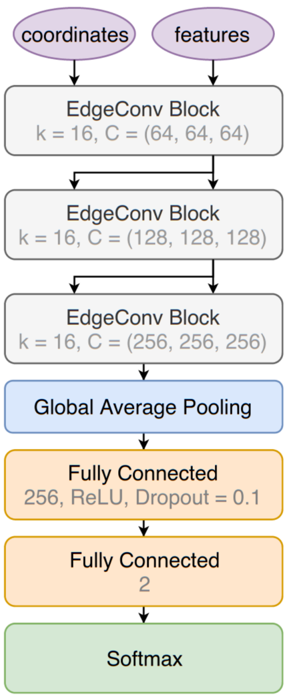


Figure 3.7.: The architecture of ParticleNet. From [27].

in a future version of OrcaNet. For the EdgeConv block, the TensorFlow implementation in [29] is used.

3.5.3. ParticleNet architecture

The ParticleNet architecture is a GNN architecture proposed and used in [27]. It is shown in fig. 3.7. ParticleNet consists of three EdgeConv blocks using $k = 16$ nearest neighbors. Each kernel network, denoted by C, consists of three layers with a constant number of neurons per layer. The number of neurons in the kernel network layers doubles after each EdgeConv block, going from 64 to 256. After the last EdgeConv block, "global average pooling" is performed, in which each of the 256 features in the nodes of the last layer is averaged over all nodes. Hence, for each graph, one obtains a fixed number of 256 nodes, which are then supplied to a fully connected layer with 256 nodes. This fully connected layer uses a ReLU activation function and a dropout with $p = 0.1$. The next layer is the output layer, consisting of two fully connected nodes with a softmax distribution.

This architecture was used slightly modified in [18, 19]. This thesis also frequently uses the basic structure of this architecture, but it will also deviate from it in later sections.

4. Production of additional Monte Carlo events

4.1. The simulation group dataset

Data-taking in KM3NeT is done using a "run-by-run" approach, where each run consists of a data-taking period of a few hours. Each run is associated with a set of regularly measured parameters describing, among others, the detector geometry and the optical properties of the water during the run. These runs are then simulated using Monte Carlo (MC) simulations, where the measured parameters are given as input to the simulation chain. This work uses Monte Carlo simulations for ORCA6, i.e., for the KM3NeT/ORCA detector consisting of six DUs. The version of the MC simulations used is primarily v7.2. Runs from v7.1 were also included if they were not present in v7.2. The data consists of 2553 runs with almost 2.5 million neutrino events. This dataset will be referred to as the "simulation group dataset" in the following.

The composition of the simulation group dataset can be seen in fig. 4.1. The left figure shows that about 400×10^3 tau neutrino events are included in the dataset, making the ν_τ^{CC} interaction type the least represented interaction type of the dataset. As the right figure of fig. 4.1 shows, the tau class represents only 16.5% of the data. Since this work focuses on training GNNs to recognize tau neutrino events in a dataset, the composition of the dataset is suboptimal for this task.

The simulation group dataset consists of neutrino events in the energy range from 1 GeV to 100 GeV. The lower bound of 1 GeV is motivated by the energy threshold of ORCA, which is about 3 GeV [16]. The upper bound of 100 GeV is motivated by the energy range in which tau neutrinos can be expected. The oscillation probabilities in fig. 1.5 show that most tau neutrinos are expected below 50 GeV. Hence, the chosen energy range from 1 GeV to 100 GeV spans the complete energy range in which oscillations into

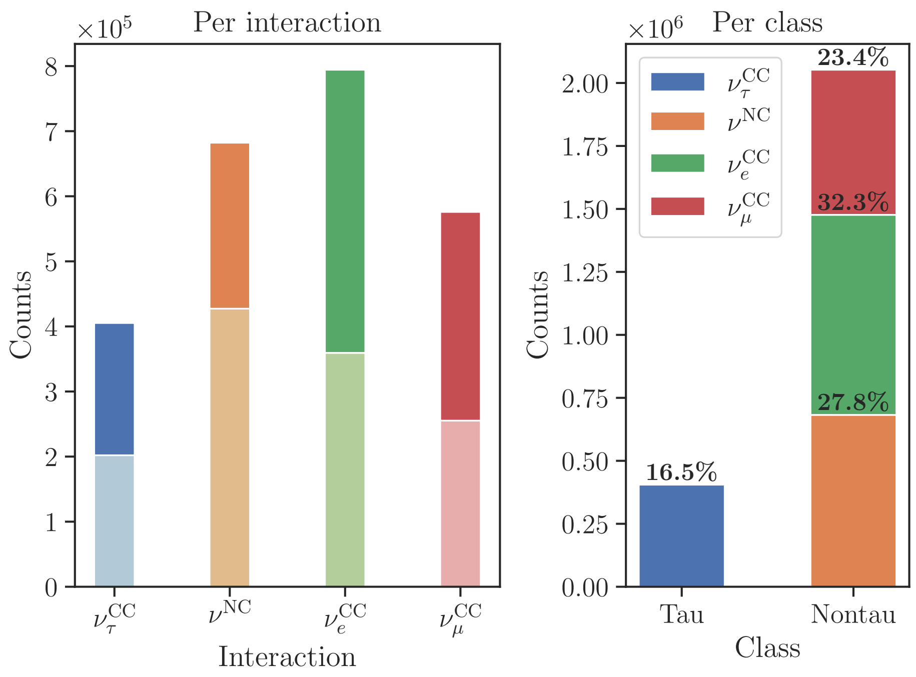


Figure 4.1.: The composition of the simulation group dataset. The left figure shows the number of neutrino events per interaction type, where the lower (upper) part of each bar shows the fraction of (anti)particle events in the bar. The right figure shows the number of events per class.

tau neutrinos can happen. The MC simulations generate neutrinos following an energy spectrum $\propto E^{-\alpha}$, where the spectral index α is chosen in the range $[2, 3]$ dependent on the neutrino interaction type. These generated neutrinos are processed through the simulation pipeline until one obtains a final set of triggered events, which is then stored and used in, e.g., the simulation group dataset defined previously. The energy spectrum of the triggered events in the simulation group dataset is shown in fig. 4.2.

The early attempts of training a GNN on this data used a reweighting scheme to achieve "balanced classes", which means that every class is represented equally in the dataset. In an imbalanced class problem, models often learn to predict the majority class disproportionately often. The origin of this behavior can be understood if one looks at the loss function defined in eq. (3.7), in which most terms $l(y, f_\Theta(x))$ in the average over the training dataset are caused by samples of the majority class. Hence, predicting the majority class for most of the samples is an effective way of minimizing the loss function. This behavior can be countered by assigning a sample weight to each

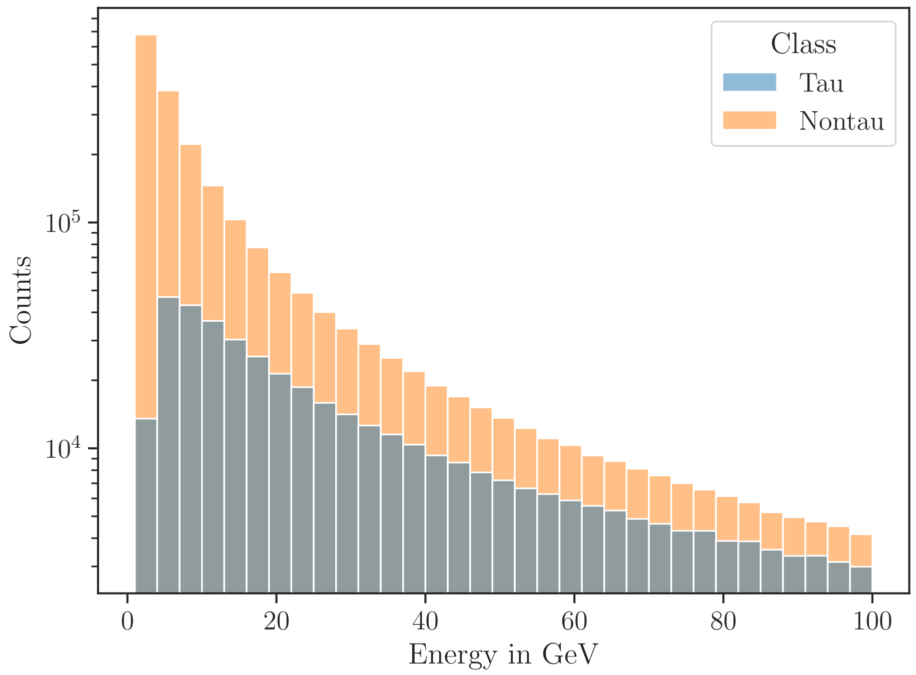


Figure 4.2.: The unweighted energy distribution of the simulation group dataset for the tau and nontau class. Note the logarithmic scale on the y-axis.

sample and replacing the average in eq. (3.7) by a weighted average, often improving the performance of machine learning models on imbalanced datasets. The weights are chosen such that the total number of events is conserved, but each class is represented equally in the dataset. Additionally, the weights were chosen such that each interaction type is represented equally in the dataset. The effective composition of the weighted dataset can be seen in fig. 4.3. This dataset is split into a training set consisting of 70% of the events and a validation set with 15% of the data. The remaining 15% of events are used as a test dataset, typically used in machine learning tasks to evaluate the final model that had the best performance on the validation set after training many models. Training many models with subsequent evaluation on the same validation data could lead to a model that performs well on the validation set by chance. Hence, a final evaluation on a previously unused test dataset is carried out to eliminate this bias.

Training on this dataset led to models that learned to base their predictions on the neutrino's energy. This effect can be seen in fig. 4.4, which shows the results of the evaluation of a model on the validation set. The model predominantly assigns lower tau scores to events with energies below 10 GeV and higher tau scores to events with higher

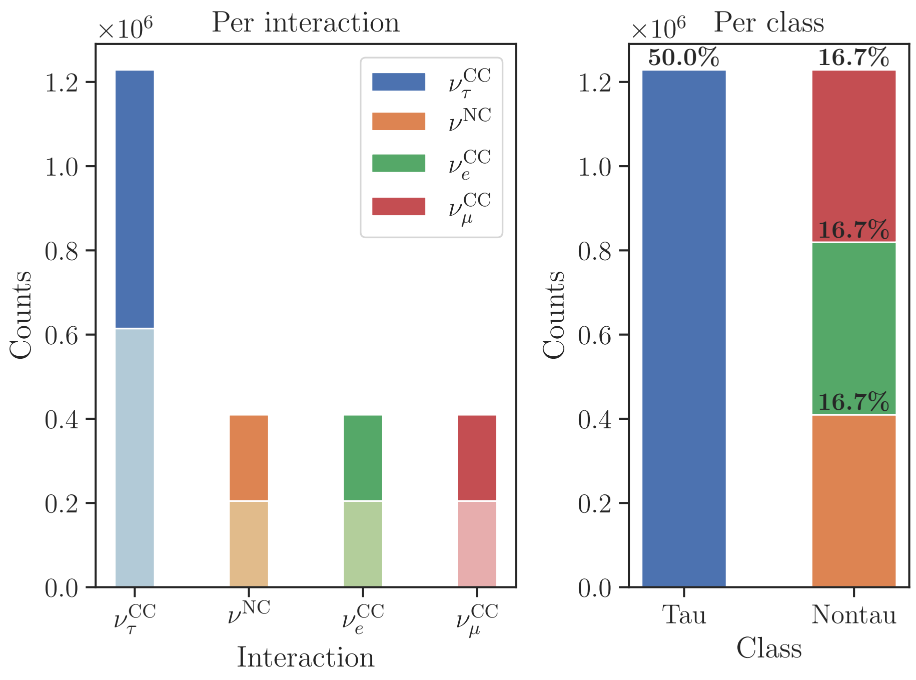


Figure 4.3.: The effective composition of the simulation group dataset with reweighting for balanced classes is shown.

energies. This behavior happens in both histograms, i.e., it is independent of the true class of the neutrino event. Atmospheric neutrinos do not have an energy spectrum like this, which would cause the GNN to fail on actual data. The origin of this behavior lies in the effective energy spectrum of the simulation group dataset with balanced class weighting, which can be seen in fig. 4.5. The distributions look different than in fig. 4.2 due to the balanced class weighting, which primarily causes an upscaling of the tau neutrino events. Most events below 10 GeV are nontau events, and most events above that energy belong to the tau class. Hence, nontau events are the majority class in the subset of all events below 10 GeV and vice versa. Following the same argumentation above, the model learns an undesired bias towards the majority class in both energy regions, leading to poor performance on actual data. Again, this effect could be countered by using a reweighting strategy, i.e., weighting the events such that the classes are balanced and the energy spectra are flat. While reweighting is effective in reducing unwanted biases to some extent, it comes to its limits if the underlying dataset has a lack of statistics in regions of the phase space.

The observation of effects like in fig. 4.4 lead to the impression that the GNN's performance

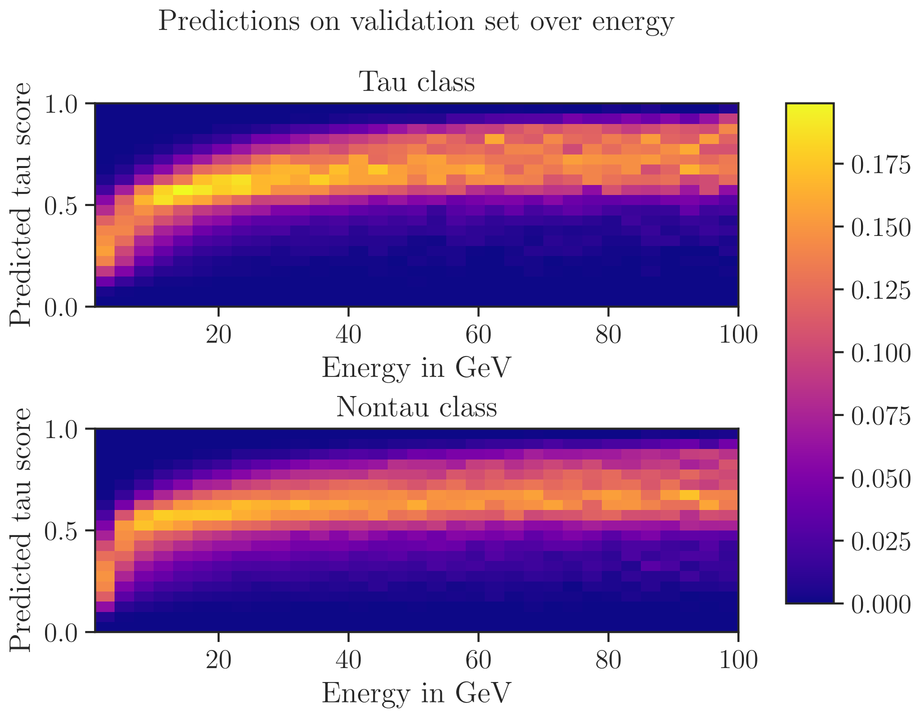


Figure 4.4.: The 2D histograms show the predicted tau score over the true energy of the neutrino. The top figure consists of events belonging to the tau class, while the bottom figure shows the nontau class. The bins are normalized along the y-axis.

would be impaired due to "being distracted" by underlying distributions in the data, e.g., by the energy spectrum. In the early steps of this thesis, it was still unclear which features of the underlying dataset could be exploited by the GNNs to maximize performance. The initial suspicion was that the main features could be the subtle differences in the event topologies shown in section 2.3. To allow the GNNs to focus on these features, one must provide a training dataset with flat distributions of quantities like the energy and the zenith angle. Hence, it was decided to produce a new MC training set to test this suspicion and to tackle the low statistics of tau events in the simulation group dataset. This new training set would consist of 50% tau events and 50% nontau events with a flat energy spectrum to reduce the GNNs' dependency on the energy as shown in fig. 4.4. Other physical quantities like the zenith distribution could be tackled by reweighting or, if necessary, explicitly flattened in a further MC production. This way, the suitability of the available training dataset for the task of tau identification could be improved step by step.

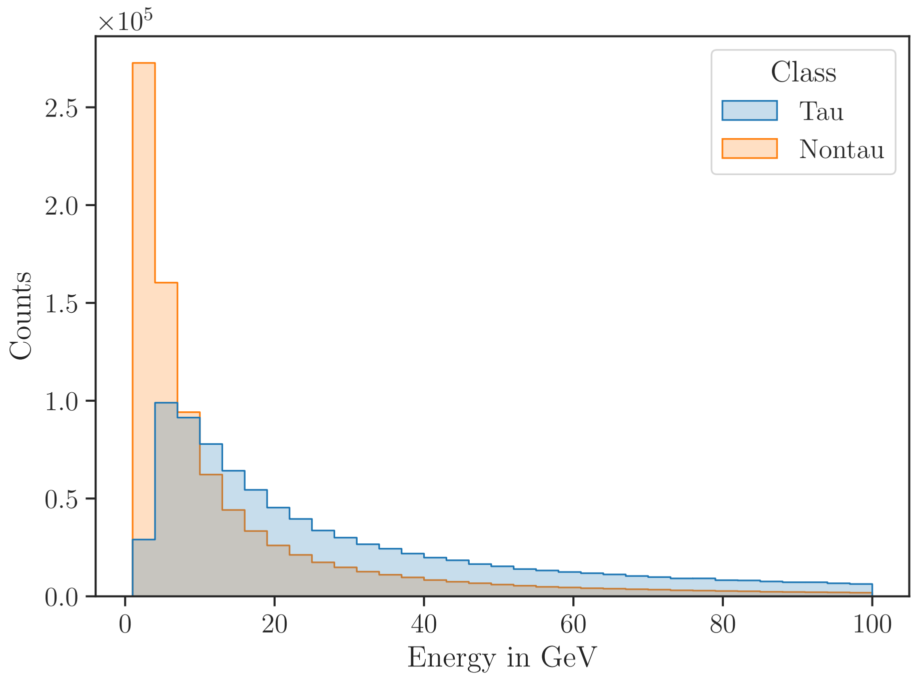


Figure 4.5.: The effective energy spectrum of the training set of the simulation group dataset with balanced class weighting is shown. Note that the y-axis has a uniform scale in contrast to fig. 4.2.

4.2. Implementation of a tool to simulate arbitrary energy spectra

The simulation chain of KM3NeT does not allow simply producing datasets of triggered events with an arbitrarily user-specified energy spectrum. Hence, it was decided to develop a small tool that allows this functionality to make the production of a dataset with a flat energy spectrum possible.

While a detailed discussion of KM3NeT's entire simulation chain is beyond the scope of this thesis, the main building blocks of the official simulation chain used for GeV neutrinos can be seen in fig. 4.6. The chain begins with the neutrino event generator "gSeaGen", which is responsible for generating a user-specified number of neutrinos n^{gen} and their subsequent interactions with the media surrounding the detector[30]. The secondary particles and the induced Cherenkov light are propagated further by "KM3Sim", which registers when a PMT of the detector is hit by light[31]. After that, "JTriggerEfficiency"

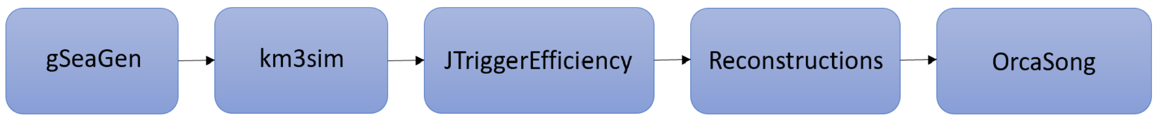


Figure 4.6.: A simplified representation of the simulation chain in KM3NeT for neutrinos in the GeV energy regime.

adds background hits caused by sources like the K^{40}-decay, simulates the efficiency of the PMTs, and applies different triggers to the simulated hits[32]. The events that fired a trigger are referred to as "triggered events". The output of JTriggerEfficiency is then given to several reconstruction algorithms adding their reconstructed quantities to the files. These files are processed further by "OrcaSong", which converts them into formats suitable for various neural network architectures[33].

The simulation chain shown in fig. 4.6 does not allow the production of a fixed number of triggered events. Instead, the user has to specify to gSeaGen that it should produce m^{gen} neutrinos that interact via interaction type ν and are sampled from an energy spectrum $\propto E^{-\alpha}$ in the range $(E_{\text{start}}, E_{\text{end}})$. The term "interaction type" is used as in section 2.3, where it specifies the flavor of the interacting neutrino and whether it interacted by a charged or neutral current interaction. Additionally, it specifies whether the interacting neutrino is a particle or an antiparticle. This discussion of the simulation chain is, of course, very simplified, but it is sufficient to understand the underlying method that the tool is using. The resulting number of triggered events m^{trig} produced by the simulation chain is a random variable depending on the input parameters specified to gSeaGen. The central idea behind the tool is to estimate the probability that a generated neutrino results in a triggered event by repeatedly executing the simulation chain, shown in detail in algorithm 1. In the beginning, the user has to choose an energy interval $(E_{\text{start}}, E_{\text{end}})$ and divide it into a set of small bins $\{(E_i, E_{i+1})\}_i$, where each energy bin is labeled by i. Note that these bins are not required to be uniformly spaced. The energy bins will be used by gSeaGen later to sample neutrinos from a flat energy spectrum, which can be achieved by specifying $\alpha = 0$. In addition, the user has to specify an interaction type ν that is supposed to be simulated by gSeaGen. The following discussion focuses on one specific combination (ν, i) of an interaction type ν and an energy bin i since the tool is implemented to execute the algorithm for each combination (ν, i) in parallel. Furthermore, the number of triggered neutrino events $n^{\text{trig}}_{\text{target}}$ of interaction type ν and

Algorithm 1 Algorithm implemented in the tool

```
Input: $\nu$, $i$, $n^{\text{trig}}_{\text{target}}$, $\Delta n^{\text{trig}}$
Output: Files containing $\approx n^{\text{trig}}_{\text{target}}$ events of interaction type $\nu$ in energy bin $i$
 1: $p \leftarrow 0.2$                                        ▷ initial value
 2: $n^{\text{gen}} \leftarrow 0$
 3: $n^{\text{trig}} \leftarrow 0$
 4: while $n^{\text{trig}} < n^{\text{trig}}_{\text{target}}$ do
 5:     $m^{\text{gen}} \leftarrow \Delta n^{\text{trig}}/p$
 6:     $m^{\text{trig}} \leftarrow$ RunSimulationChain($\nu$, $i$, $m^{\text{gen}}$)
 7:     $n^{\text{gen}} \leftarrow n^{\text{gen}} + m^{\text{gen}}$
 8:     $n^{\text{trig}} \leftarrow n^{\text{trig}} + m^{\text{trig}}$
 9:     $p \leftarrow n^{\text{trig}}/n^{\text{gen}}$
10: end while
11: return

12: function RunSimulationChain($\nu$, $i$, $m^{\text{gen}}$)
13:     generate $m^{\text{gen}}$ neutrinos sampled from flat energy spectrum in bin $i$
14:     simulate interactions via interaction type $\nu$
15:     run rest of the pipeline
16:     return number of triggered events
17: end function
```

in energy bin i that should be produced by the tool has to be specified. The last input that the tool requires is the quantity Δn^{trig}, which is the number of triggered events that the tool tries to produce in each iteration. The number of neutrinos m^{gen} that need to be generated to obtain approximately Δn^{trig} in that iteration is obtained by using the estimated probability p that a generated neutrino results in a triggered event. This probability is approximated using the total number of generated neutrinos n^{gen} and triggered events n^{trig} so far. If the total number of produced triggered events n^{trig} exceeds the specified $n^{\text{trig}}_{\text{target}}$, the algorithm terminates.

The tool is implemented in the workflow management system “Snakemake”[34], which allows the implementation and execution of software pipelines in an easy and parallelized way. Each building block of the official simulation chain shown in fig. 4.6 is implemented as a rule in Snakemake. These rules use the official singularity containers and configuration files provided by the simulation group to ensure proper functioning. Each rule defines how to create a set of output files from a set of input files. If the user specifies that a particular file should be created, Snakemake automatically searches for the chain of rules that has to be executed to obtain that file. Rules in this chain not depending on each other are executed in parallel, which makes effective utilization of resources on large computing clusters possible. The tool also provides two “profiles” named “local” and “slurm”, which can be used to run the tool quickly on either the local machine or by sending out the rules as jobs to a computing cluster using the workload management system “Slurm”[35]. The tool’s code and documentation can be found in [36].

Currently, the tool has two limitations. The first is that it is only implemented for the pipeline shown in fig. 4.6, which is suitable for neutrinos in the GeV energy range. If one wants to use, e.g., a different light propagator than KM3Sim, the corresponding rule for that would have to be implemented in Snakemake. The second limitation is that the tool only simulates events for one specific run. The reason is that the tool was built starting from a simple case and extended further to more complexity in a step-by-step approach. Since implementing a proper run-by-run simulation is a complex task, its implementation was postponed to a later step. Hence, the project was started by simulating one particular, randomly chosen run, and it was subsequently checked if the resulting simulations would already be sufficient to improve the performance of the GNNs. In this study, the run with ID 8007 was used for the simulations. As shown later in section 5.2, the resulting simulations are indeed sufficient to enhance the GNNs’ performance.

Nevertheless, extending the tool to perform run-by-run simulations could benefit future machine learning studies. A few weeks after the implementation of the tool was started, efforts began to implement KM3NeT’s complete run-by-run simulation pipeline in Snakemake[37]. One way to extend this tool could be to switch out the currently implemented special-case simulation pipeline with this more general pipeline.

4.3. The flat dataset

The tool described in the last section was used to produce a new dataset for tau identification. Due to its flat energy spectrum, this dataset will be called the “flat dataset” from now on. The composition of the flat dataset is shown in fig. 4.7. The flat

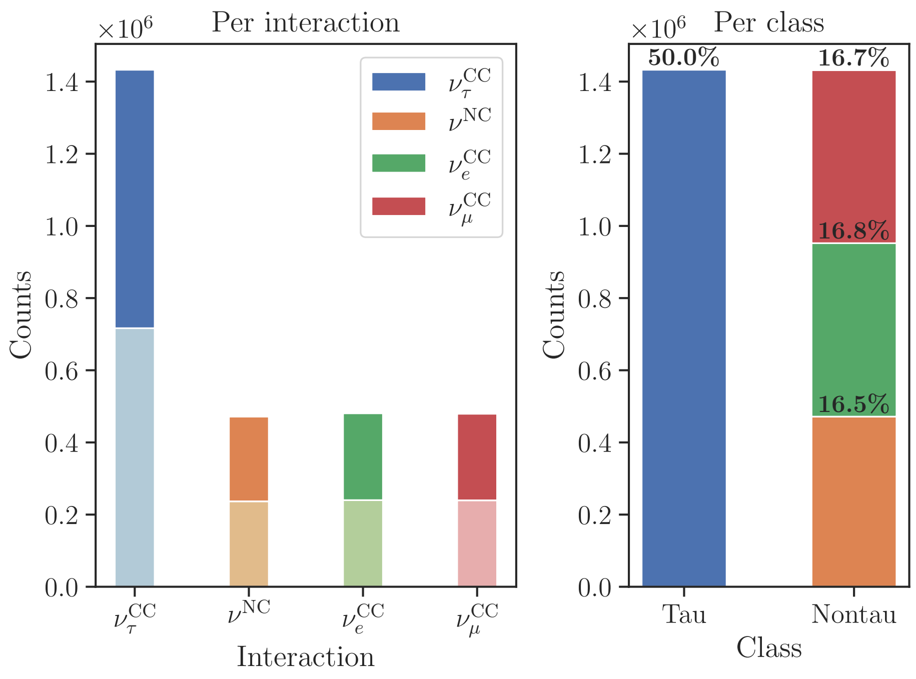


Figure 4.7.: The composition of the flat dataset, in which the classes and interaction types are balanced.

dataset has balanced classes and interaction types without relying on reweighting. It consists of about 2.8 million neutrino events, which can be split into a training set of about 2 million events and a validation and test set of 400 000 events each. Although this validation and test set was used during the studies in the following sections, the final evaluation is always done on the validation and test dataset of the simulation group dataset since the latter dataset follows the run-by-run approach.

The flat dataset has a nearly flat energy spectrum that is added to the appendix in fig. A.1. The only bin that deviates visibly from the others is the lowest energy bin with events in the energy range from 1 GeV to 4 GeV. The probability that a generated interaction results in a triggered event is tiny for these low-energy neutrinos, meaning they need much more computation time to obtain a fixed number of triggered events. This behavior is intuitive since lower-energy neutrinos have a smaller interaction cross-section than higher-energy neutrinos and low-energy interactions produce less light, reducing the probability that the event fires a trigger. Since the height differences of the first bin compared to the other bins are not too significant, it was decided to counter this effect by reweighting. As shown later in section 5.2, the height difference of the bin will become

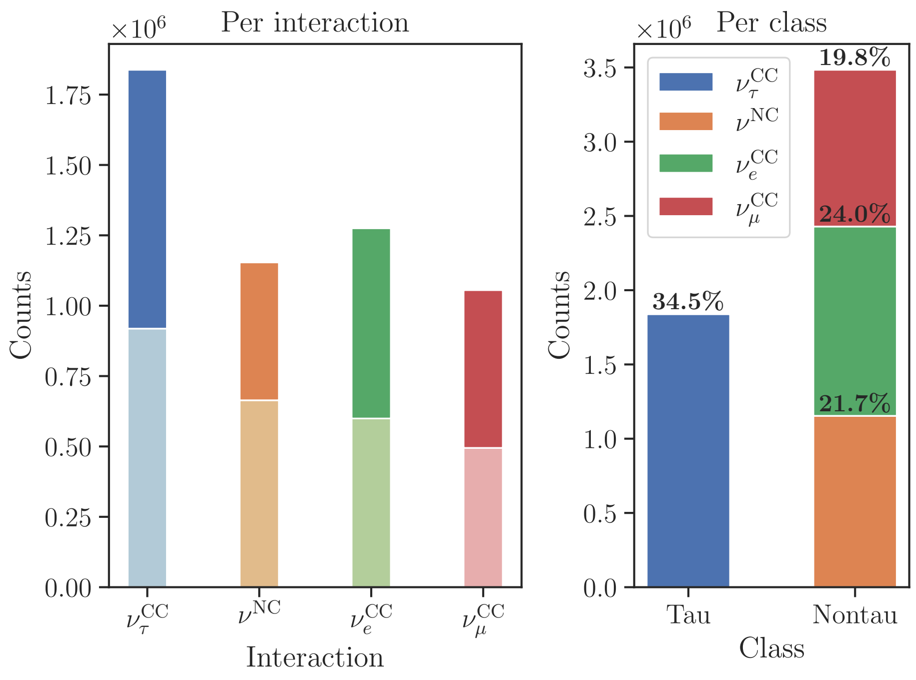


Figure 4.8.: The composition of the merged dataset.

irrelevant once an optimized reweighting strategy is applied.

4.4. The merged dataset

As discussed in section 4.2, the flat dataset was not produced following a run-by-run approach but by simulating only one particular run. This limitation leads to discrepancies between the flat dataset and the simulation group dataset that could impair the GNNs' performance if they are trained on the flat dataset but validated and tested on the simulation group dataset. Hence, to counter these discrepancies while still improving the tau statistics of the dataset, a third dataset is tested that merges the simulation group dataset and the flat dataset. From now on, this dataset will be called the "merged dataset". The composition of the merged dataset can be seen in fig. 4.8. The ν_τ^{CC} interaction type is this dataset's most represented interaction type. While the tau class is still the minority class in the dataset, its share in the dataset is increased to 34.5%, which is a considerable improvement of tau statistics compared to the share of 16.5% in the simulation group dataset shown in fig. 4.1.

5. Manual optimizations

5.1. Discussion of reweighting methods

5.1.1. Honda weighting

As discussed in section 1.4, atmospheric neutrinos are created due to interactions between cosmic rays and nuclei in the atmosphere. The resulting atmospheric neutrino flux is modeled and simulated by the HKKM group[38], which published flux tables calculated at different detector sites[39]. Different tables were calculated for different solar activities and times of the year, including seasonal variations of the atmospheric neutrino flux due to variations in the atmosphere's density. The most specific table includes the energy dependence of the flux and the dependence on the zenith and the azimuth. In the case of studies with the KM3NeT/ORCA detector, the azimuth-averaged table for the Frejus site can be used, which is located close to the KM3NeT/ORCA detector[14]. Specifically, this thesis uses the time- and azimuth-averaged flux at the Frejus site without the mountain over the detector and at a minimum of solar activity. This flux will be denoted as $\phi_\alpha^{\mathrm{H}}(E, \cos\theta)$, where the H stands for "Honda", $\alpha \in \{e, \mu\}$ denotes the flavor of the neutrinos in the flux, and E and θ denote the energy and zenith angle. Note that $\phi_\alpha^{\mathrm{H}}(E, \cos\theta)$ only exists for electron and muon neutrinos since tau neutrinos are not produced in the atmosphere. The flux table is accessed via "km3flux", which offers functionality to access a collection of atmospheric neutrino flux tables[40]. The flux table contains information for particles as well as antiparticles. Only particles are discussed in the following for brevity, but the definitions and calculations apply analogously to antiparticles.

As discussed in section 1.2, a neutrino produced in the flavor state ν_α can oscillate into a different flavor ν_γ during its propagation, which is the reason why tau neutrinos can be found in the atmospheric neutrino flux. Since the Honda flux $\phi_\alpha^{\mathrm{H}}(E, \cos\theta)$ only describes the flux of muon and electron neutrinos produced in the atmosphere, the probabilities for neutrino oscillations have to be taken into account to obtain the expected flux at the detector site. The energy- and zenith-dependent probability $P_{\alpha\to\gamma}(E, \cos\theta)$ that a neutrino produced in the atmosphere with flavor ν_α oscillates into flavor ν_γ can be calculated with "OscProb"[41], which can be accessed, e.g., via "km3services"[42]. The expected flux at the detector, $\phi^{\mathrm{D}}_{\gamma}(E, \cos\theta)$, can be calculated by

$$\phi^{\mathrm{D}}_{\gamma}(E, \cos\theta) = P_{e\to\gamma}(E, \cos\theta)\,\phi^{\mathrm{H}}_{e}(E, \cos\theta) + P_{\mu\to\gamma}(E, \cos\theta)\,\phi^{\mathrm{H}}_{\mu}(E, \cos\theta), \tag{5.1}$$

where $\gamma \in \{e, \mu, \tau\}$. Note that $\phi^{\mathrm{D}}_{\gamma}(E, \cos\theta)$ has a nonzero flux for tau neutrinos since the electron and muon neutrinos produced in the atmosphere can oscillate into tau neutrinos.

The Honda flux can be used to reweight neutrino events in a dataset such that its composition approximates the composition of measured data. Since the GNNs trained in this thesis should be applied to measured data in future analyses, they are evaluated on validation and test datasets weighted according to the Honda flux, which will be referred to as being "Honda weighted" from now on.

The following calculations show how the Honda weight $w^{\mathrm{H}}_{r,\gamma,k}$ for a neutrino event $e_{r,\gamma,k}$ can be calculated, where r is the run ID of the run to which the event belongs, γ denotes the interaction type $\nu^{\mathrm{CC}}_{\gamma}$, and k labels the particular event. Note that, for now, only charged current interactions are discussed. The notation is inspired by the structure of run-by-run MC simulations in KM3NeT/ORCA. For each run $r \in \{1, 2, \ldots, R\}$, each interaction type γ is simulated independently, resulting in one file for each tuple (r, γ). Each of these files contains $N_{r,\gamma}$ triggered events, where each event is labeled by the tuple (r, γ, k) with $k \in \{1, \ldots, N_{r,\gamma}\}$. Hence, a particular event can be denoted as $e_{r,\gamma,k}$. The neutrino event generator gSeaGen provides two quantities that can be used to calculate the Honda weight, namely the number of generated neutrinos $n^{\mathrm{gen}}_{r,\gamma}$ per file (r, γ) and the $w_{2,r,\gamma,k}$ weight for each event $e_{r,\gamma,k}$. A detailed definition of the w_2 weight can be found in [30]. It can be understood naively as

$$\frac{w_{2,r,\gamma,k}}{n^{\mathrm{gen}}_{r,\gamma}} = \frac{1}{\phi^{\mathrm{Gen}}_{r,\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k}) \cdot 1\,\mathrm{s}}, \tag{5.2}$$

where $\phi^{\mathrm{Gen}}_{r,\gamma}(E, \cos\theta)$ is the neutrino flux that gSeaGen generated to obtain the simulated events in file (r, γ). Note that $\phi^{\mathrm{Gen}}_{r,\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k})$ is multiplied by 1 s in the denominator of eq. (5.2) to cancel out a dimension of time in the next equation. The Honda weight

$w^{\mathrm{H}}_{r,\gamma,k}$ is defined as

$$w^{\mathrm{H}}_{r,\gamma,k} = \frac{\phi^{\mathrm{D}}_{\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k})}{\phi^{\mathrm{Gen}}_{r,\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k})} \cdot \frac{t_r}{1\,\mathrm{s}} \tag{5.3}$$

$$= \frac{w_{2,r,\gamma,k}}{n^{\mathrm{gen}}_{r,\gamma}} \cdot \phi^{\mathrm{D}}_{\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k}) \cdot \frac{t_r}{1\,\mathrm{s}} \tag{5.4}$$

where t_r is defined as the livetime of run r, i.e., the time the detector collected data during the run r. The components of eq. (5.3) can be understood intuitively: if the physically expected flux at the detector $\phi^{\mathrm{D}}_{\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k})$ is, e.g., double as large as the flux $\phi^{\mathrm{Gen}}_{r,\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k})$ used during the generation of the dataset, the event should be scaled up by a factor of two. Similarly, if the livetime t_r of run r is double as long as the livetime $t_{r'}$ of run r', the Honda weight of the events in run r should be scaled up by a factor of two compared to the events in run r' since run r took twice as much data as run r'.

Instead of looking up the actual livetime of each run $r \in \{1, 2, \ldots, R\}$, the livetime was approximated to be equal for each run, i.e., $t_r = t \quad \forall r \in \{1, 2, \ldots, R\}$. This approximation is justified since the flavor composition of the resulting dataset will not be affected by changes to the livetime t_r affecting all flavors (r, γ) equally. Additionally, it should be noted that the livetime t can be chosen arbitrarily since it will only scale the resulting Honda weights, but it will not change their relative size. Hence, it was decided to choose t such that the effective total number of events is conserved, which can be achieved by defining

$$t = N \cdot \left(\sum_{r,\gamma,k} \frac{w_{2,r,\gamma,k}}{n^{\mathrm{gen}}_{r,\gamma}} \cdot \phi^{\mathrm{D}}_{\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k}) \right)^{-1} \cdot 1\,\mathrm{s}, \tag{5.5}$$

where $N = \sum_{r,\gamma} N_{r,\gamma}$ is the total number of neutrino events in the dataset.

So far, it was assumed that each run has MC simulations for each charged current interaction type. As it turns out, this is not true since some simulated runs have missing files. If a run r has a missing file $(r, \hat{\gamma})$, adding the run to the dataset would cause an underrepresentation of the interaction type $\hat{\gamma}$ in the dataset. Hence, the weights of events belonging to the interaction type $\hat{\gamma}$ must be scaled up. This upscaling can be done by using the total number of runs R_{γ} existing for interaction type γ, which changes eqs. (5.4) and (5.5) to

$$w^{\mathrm{H}}_{r,\gamma,k} = \frac{w_{2,r,\gamma,k}}{n^{\mathrm{gen}}_{r,\gamma}} \cdot \phi^{\mathrm{D}}_{\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k}) \cdot \frac{t}{R_\gamma \cdot 1\,\mathrm{s}} \tag{5.6}$$

$$t = N \cdot \left(\sum_{r,\gamma,k} \frac{w_{2,r,\gamma,k}}{n^{\mathrm{gen}}_{r,\gamma} \cdot R_\gamma} \cdot \phi^{\mathrm{D}}_{\gamma}(E_{r,\gamma,k}, \cos\theta_{r,\gamma,k}) \right)^{-1} \cdot 1\,\mathrm{s}, \tag{5.7}$$

where eq. (5.6) can be interpreted as dividing the livetime t by the number of runs R_γ for interaction type γ. If a particular interaction type γ has more runs than another interaction type γ', this means that each run of interaction type γ receives less livetime than each run of interaction type γ', such that all the runs of interaction type γ have a total livetime t that is equal to the total livetime t of all the runs of interaction type γ'. Note that eq. (5.6) defines Honda weighting only for charged current interactions. The situation is more complicated for ν^{NC} interactions. The cross-section for a neutral current interaction is independent of the neutrino flavor, which means that it would be redundant to simulate $\nu^{\mathrm{NC}}_{\alpha}$ for all three neutrino flavors $\alpha \in \{e, \mu, \tau\}$. Hence, only the interaction type ν^{NC}_{μ} is simulated by convention. From now on, neutral current interactions will be denoted by having $\gamma = \nu$ in contrast to charged current interactions with $\gamma \in \{e, \mu, \tau\}$. For the rest of this thesis, the definition of Honda weighting for neutral current interactions is

$$w^{\mathrm{H}}_{r,\nu,i} = \frac{w_{2,r,\nu,k}}{n^{\mathrm{gen}}_{r,\nu}} \cdot \phi^{\mathrm{D}}_{\mu}(E_{r,\nu,k}, \cos\theta_{r,\nu,k}) \cdot \frac{t}{R_\nu \cdot 1\,\mathrm{s}} \tag{5.8}$$

$$t = N \cdot \left(\sum_{r,\nu,k} \frac{w_{2,r,\nu,k}}{n^{\mathrm{gen}}_{r,\nu} \cdot R_\nu} \cdot \phi^{\mathrm{D}}_{\mu}(E_{r,\nu,k}, \cos\theta_{r,\nu,k}) \right)^{-1} \cdot 1\,\mathrm{s}, \tag{5.9}$$

which is exactly the same definition as eqs. (5.5) and (5.6) with $\gamma = \nu$ *except* for the expected flux at the detector $\phi^{\mathrm{D}}_{\mu}(E_{r,\nu,k}, \theta_{r,\nu,k})$, which still has $\gamma = \mu$. This circumstance is caused by a bug in the implementation of the Honda weighting, which was found after the trainings in the following chapters were already finished. The correct Honda weighting for neutral current interactions would use the expected flux at the detector

$$\phi^{\mathrm{D}}_{\nu}(E, \cos\theta) = \phi^{\mathrm{H}}_{e}(E, \cos\theta) + \phi^{\mathrm{H}}_{\mu}(E, \cos\theta), \tag{5.10}$$

since the neutral current interactions can be induced by neutrinos of all flavors. This bug causes an underrepresentation of neutral current interactions in the dataset, as seen in fig. 5.1. The right figure shows a nontau class in which the fraction of neutral current interactions is 7.6%, which is larger than the fraction of 5.0% in the left figure.

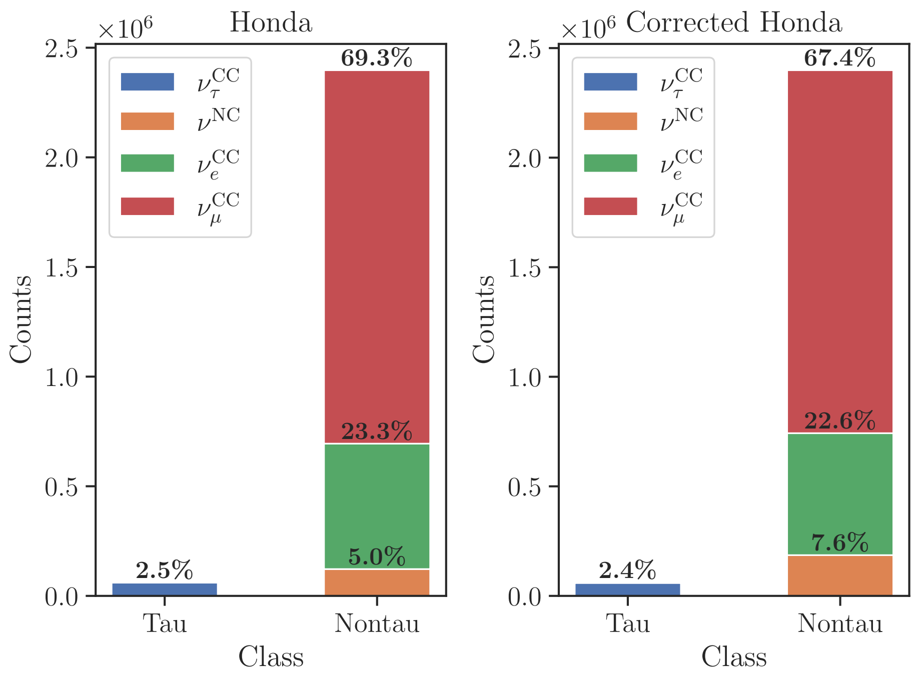


Figure 5.1.: The composition of the tau and nontau class for the simulation group dataset. The left figure applies Honda weighting to the events, while the right figure applies the corrected Honda weighting.

The corrected implementation of Honda weighting for neutral current interactions will only be used at the end of the thesis to evaluate the best-performing GNN. Hence, if not stated otherwise, eqs. (5.8) and (5.9) will be the definition for Honda weighting for neutral current interactions. Figure 5.2 shows the energy spectra of the Honda weighted simulation group dataset. The figure shows that the energy spectrum of the triggered tau events has a maximum at around 16.5 GeV, while the maximum of the nontau class is much lower at around 5.5 GeV. The GNNs could use information like that to separate neutrino events belonging to the tau class from nontau events. Additionally, the $\cos\theta$ distributions of the Honda weighted simulation group dataset is shown in fig. 5.3, where θ is the zenith angle of the neutrino that caused the neutrino event. The figure shows that almost all tau neutrino events are upgoing. This effect exists because upgoing atmospheric neutrinos had to propagate a longer distance than downgoing neutrinos since they had to travel through parts of the Earth to reach the detector. As indicated in fig. 1.5, atmospheric neutrinos in the GeV energy range need this additional propagation distance in order to be able to oscillate with a high probability into a tau neutrino.

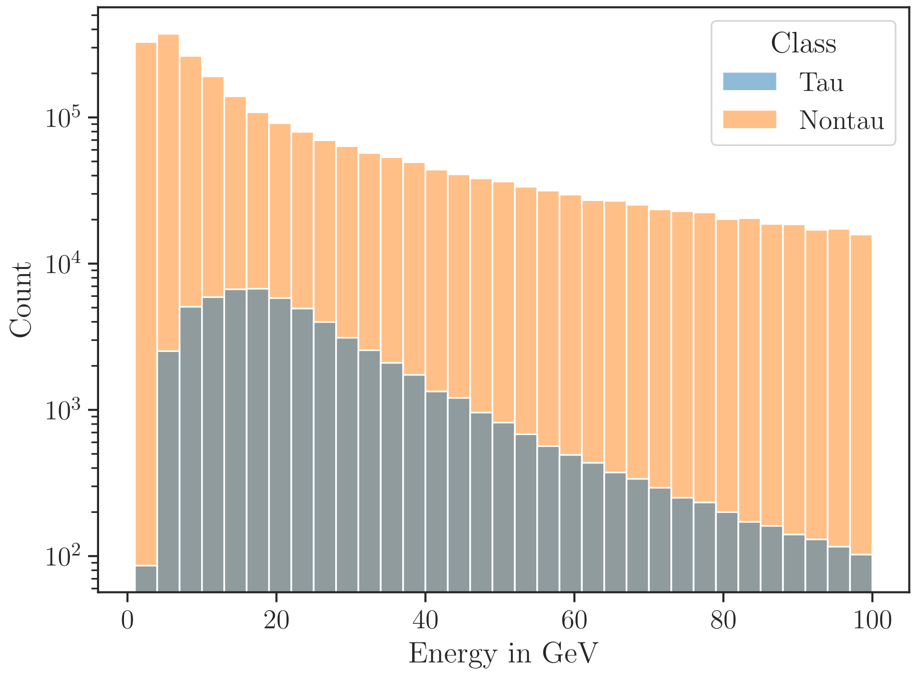


Figure 5.2.: The energy spectrum of the Honda weighted simulation group dataset.

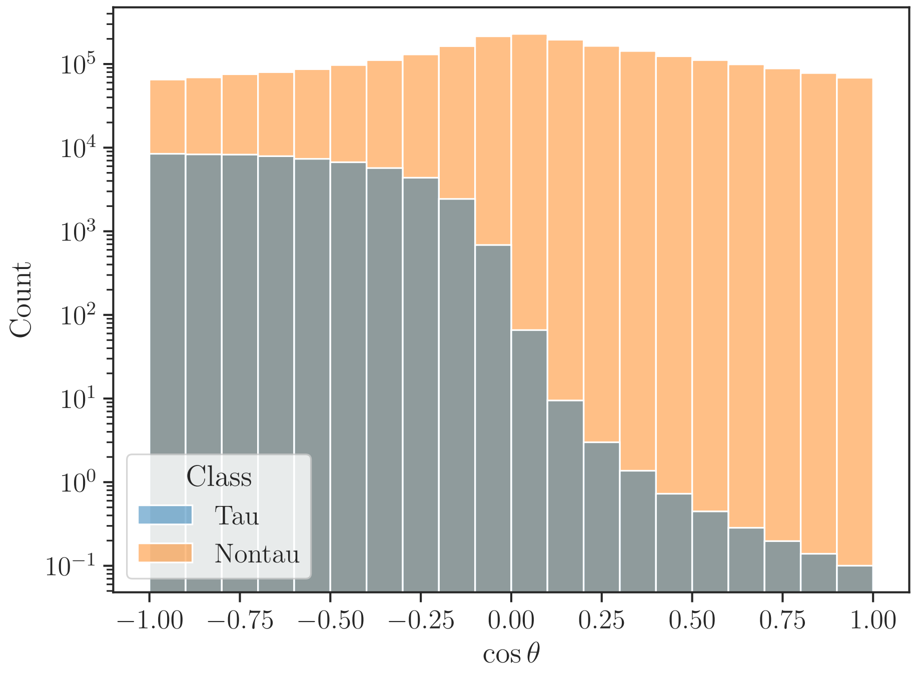


Figure 5.3.: The $\cos\theta$ distributions of the Honda weighted simulation group dataset. The value $\cos\theta = -1$ (1) corresponds to upgoing (downgoing) events.

Hence, the GNNs could use this feature to filter out many nontau events by learning that all downgoing events are nontau events. Nontau events, on the other hand, have their maximum close to the horizon. To understand this behavior, one has to remember that many atmospheric neutrinos are produced by the decay of atmospheric muons as shown in eqs. (1.14) and (1.15). Since atmospheric muons close to the horizon could travel longer distances through the atmosphere, a larger fraction of them could decay before being absorbed by the Earth.

5.1.2. Flat weighting

As discussed in section 4.1, it was not clear at the beginning of these studies which features should be used by the GNNs to reach a good performance in tau identification. The initial suspicion for suitable features were the subtle differences in the neutrino event topologies shown in fig. 2.4. The network could learn these if one supplies a dataset in which information about the energy spectrum of the data and the zenith distribution are washed out by explicitly flattening the distributions. Additionally, to achieve balanced classes and even balanced interaction types, the neutrino events are weighted to a composition as shown in fig. 4.3. From now on, this reweighting method will be called "flat weighting".

For flat weighting, one has to choose a set of energy bins $B_\mathrm{E} = \{[E_i, E_{i+1}) \mid i = 1, \dots, n_E\}$ and a set of $\cos\theta$ bins $B_{\cos\theta} = \{[\cos\theta_j, \cos\theta_{j+1}) \mid j = 1, \dots, n_{\cos\theta}\}$. In this thesis, the energy range from 1 GeV to 100 GeV was divided into $n_E = 33$ bins with a binwidth of 3 GeV each. The $\cos\theta$ range from -1 to 1 was divided into $n_{\cos\theta} = 20$ bins with a binwidth of 0.1 each. These bins were also used in the histograms shown in the above-mentioned figs. 5.2 and 5.3. Together, the energy bins B_E and the $\cos\theta$ bins $B_{\cos\theta}$ form 2D bins b_{ij} defined as

$$b_{ij} = \Big([E_i, E_{i+1}), [\cos\theta_j, \cos\theta_{j+1})\Big) \in B_E \times B_{\cos\theta}. \tag{5.11}$$

Furthermore, it will be defined that a tuple $(E, \cos\theta) \in b_{ij}$ iff $E \in [E_i, E_{i+1})$ and $\cos\theta \in [\cos\theta_j, \cos\theta_{j+1})$. The number of events in the dataset that are elements of bin b_{ij} will be denoted as N_{ij}. Analogous to the case of Honda weighting, the flat weights are normalized such that the total number of events is conserved, which means that

$$N_B = \frac{N}{n_E \cdot n_{\cos\theta}} \tag{5.12}$$

events have to be filled in each 2D bin b_{ij} to obtain both a flat energy spectrum and a flat $\cos\theta$ distribution, where N is the total number of events in the dataset. In each of these bins, half of the events should belong to the tau class and the other half to the nontau class. Furthermore, the interaction types shall be balanced in each class. This balancing can be achieved by introducing a factor f_γ, with $\gamma \in \{e, \mu, \tau, \nu, \bar{e}, \bar{\mu}, \bar{\tau}, \bar{\nu}\}$ denoting the interaction type, where $\gamma = \nu$ represents the neutral current interactions (analogous to section 5.1.1) and a bar over the interaction type denotes the antiparticle interaction. To achieve balanced interaction types as shown in fig. 4.3, f_γ has to be defined as

$$f_\gamma = \frac{1}{2} \cdot \frac{1}{2} \cdot \begin{cases} 1 & \gamma \in \{\tau, \bar{\tau}\} \\ \frac{1}{3} & \text{otherwise.} \end{cases} \tag{5.13}$$

The first factor $\frac{1}{2}$ is the result of dividing each 2D bin b_{ij} equally between the tau and the nontau class, while the second factor $\frac{1}{2}$ comes from dividing each part of the bin again in an equal manner between particle and antiparticle interaction types. Finally, the part of the bin belonging to tau events does not have to be split further, while the part of the nontau bin has to be split equally into three parts to give an equal fraction of the bin to the three nontau interaction types ν_e^{CC}, ν_μ^{CC}, and ν^{NC}. Analogous to section 5.1.1, an event $e_{r,\gamma,k}$ will be labeled by its run ID r, its interaction type γ, and by k which refers to the kth event in file (r, γ). The flat weight $w^{\mathrm{F}}_{r,\gamma,k}$ of event $e_{r,\gamma,k}$ is defined as

$$w^{\mathrm{F}}_{r,\gamma,k} = \frac{N_B \cdot f_\gamma}{N_{ij}} \quad \text{such that} \quad (E_{r,\gamma,k}, \cos\theta_{r,\gamma,k}) \in b_{ij}. \tag{5.14}$$

This definition can be understood in two steps: the first step is dividing each event in bin b_{ij} by N_{ij}, which leads to a bin b_{ij} with a height normalized to one. Since each bin should contain $N_B \cdot f_\gamma$ events of interaction type γ, each event belonging to that interaction type is scaled up by the factor $N_B \cdot f_\gamma$, resulting in a bin with a total height N_B after summing over all interaction types γ. Since each bin b_{ij} has the same height, the energy spectrum and the $\cos\theta$ distribution are flattened. The only exception for a bin not filled with N_B events is a bin b_{ij} that does not contain events of interaction type γ. Since the flat weights $w^{\mathrm{F}}_{r,\gamma,k}$ defined in eq. (5.14) are only calculated for each existing event, the fraction of bin b_{ij} belonging to events of interaction type γ will remain unfilled.

The composition of the flat weighted simulation group dataset can be seen in fig. 5.4, which shows that the classes and interaction types are balanced with a slight underrepresentation of the ν^{NC} interaction type. As discussed above, this happened because there exist some bins that do not have any event with interaction type ν^{NC} or $\bar{\nu}^{\mathrm{NC}}$. The same effect can

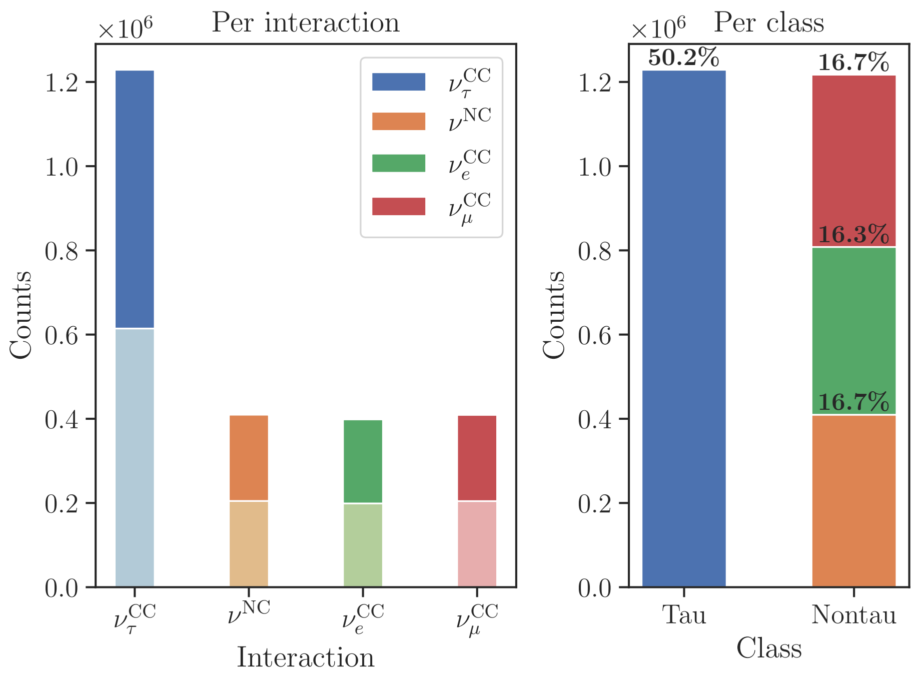


Figure 5.4.: The composition of the simulation group dataset with flat weighting.

be seen for the energy spectrum in fig. A.2 and the $\cos\theta$ distributions in fig. A.3, where the height of the bins can deviate slightly from each other if some bins are not entirely filled. Nevertheless, the differences are small enough for the context of machine learning that they should not affect the training significantly.

5.1.3. Intermediate weighting

This section discusses intermediate weighting, which is an interpolation between Honda and flat weighting. This interpolation is used in the next section to analyze whether one should use weighting emphasizing physical information about the atmospheric neutrino flux or the subtle differences in the neutrino event topologies to achieve a good performance of the GNNs in tau identification. Intermediate weighting introduces two parameters $\alpha \in [0, 1]$ and $\beta \in [0, 1]$. The parameter α determines how similar the weighting is to either the extreme cases of flat weighting ($\alpha = 0$) or Honda weighting ($\alpha = 1$). On the other hand, β determines the fraction of tau events in the dataset, which makes it possible to analyze how an over- or undersampling of tau events affects the GNNs' performance. Additionally, the suspicion that balanced classes lead to good performance can be tested, corresponding to the case $\beta = 0.5$. Let $\mathcal{T} = \{\tau, \bar{\tau}\}$ be the class of tau events and $\mathcal{T}^{\mathrm{C}} = \{e, \mu, \nu, \bar{e}, \bar{\mu}, \bar{\nu}\}$ the complementary set of nontau events, where ν represents again the neutral current interactions. The first step in intermediate weighting is to calculate the weights $w^{\mathrm{X},\beta}_{r,\gamma,k}$, which are scaled versions of the Honda weights ($\mathrm{X} = \mathrm{H}$) or the flat weights ($\mathrm{X} = \mathrm{F}$). More specifically, they are scaled such that $\beta \cdot N$ events in the total dataset with N events are tau events, implying that the remaining $(1 - \beta) \cdot N$ events are nontau events. This composition can be achieved by defining

$$w^{\mathrm{X},\beta}_{r\gamma k} = w^{\mathrm{X}}_{r\gamma k} \cdot N \cdot \begin{cases} \left(\sum_{r,k} \sum_{\gamma\in\mathcal{T}} w^{\mathrm{X}}_{r\gamma k}\right)^{-1} \cdot \beta & \text{if} \quad \gamma \in \mathcal{T} \\ \left(\sum_{r,k} \sum_{\gamma\in\mathcal{T}^{\mathrm{C}}} w^{\mathrm{X}}_{r\gamma k}\right)^{-1} \cdot (1-\beta) & \text{if} \quad \gamma \in \mathcal{T}^{\mathrm{C}} \end{cases} \tag{5.15}$$

where the same strategy as before was used to normalize the weights with the denominator and subsequently scale them up with the nominator. These scaled Honda and flat weights are then linearly interpolated to obtain the intermediate weight $w^{\mathrm{I},\alpha,\beta}_{r,\gamma,k}$ defined by

$$w^{\mathrm{I},\alpha,\beta}_{r,\gamma,k} = \alpha \cdot w^{\mathrm{H},\beta}_{r\gamma k} + (1-\alpha)\, w^{\mathrm{F},\beta}_{r\gamma k}, \tag{5.16}$$

where the parameter α controls how much the dataset deviates from flat weighting and how much it aligns with Honda weighting. The special case of $\alpha = 0$ corresponds to flat weighting scaled with β, while the case of $\alpha = 1$ corresponds to Honda weighting scaled with β, which makes it possible to use, e.g., Honda weighting ($\alpha = 1$) while still preserving balanced classes ($\beta = 0.5$).

Figure 5.5 shows how a variation in α affects the energy spectrum of the merged dataset in which class balance is ensured by fixing $\beta = 0.5$. The $\alpha = 0$ case corresponds to flat weighting, which can be confirmed by the flat blue line in fig. 5.5. For increasing α, the energy spectra follow the Honda weighting more closely, which is the $\alpha = 1$ case. The same effects can be seen in fig. 5.6, where the $\cos\theta$ distribution is shown for varying α. The left plot in the figure shows that the strong suppression of downgoing tau events becomes effective only for large values of α.

5.2. Optimizing the weighted training set

This section uses the three datasets defined in chapter 4 and the reweighting methods defined in section 5.1 to search for the optimal training dataset for tau identification, which can then be used during the automatic hyperparameter optimization in the following

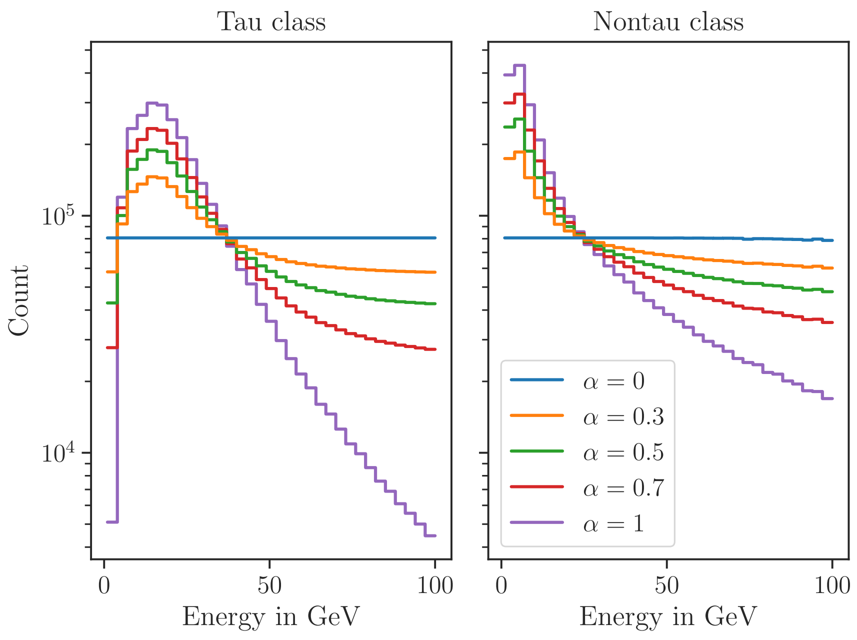


Figure 5.5.: The effects of a variation in α on the energy spectrum of the merged dataset. The parameter β is fixed to 0.5.

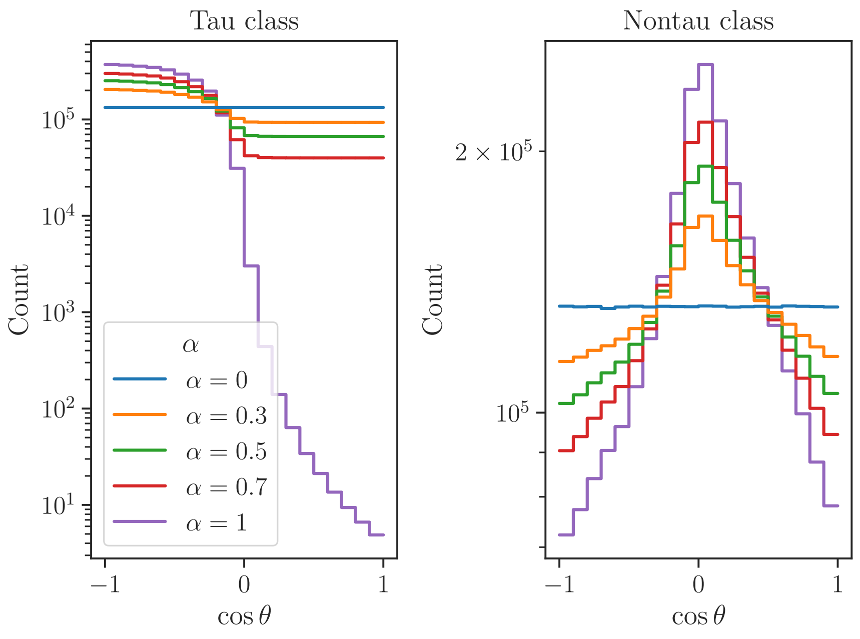


Figure 5.6.: The effects of a variation in α on the $\cos\theta$ distribution of the merged dataset. The parameter β is fixed to 0.5.

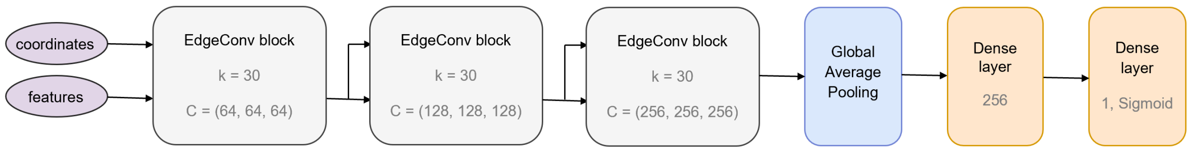


Figure 5.7.: The GNN used for the optimization of the training dataset.

chapter. The training, validation, and test sets used throughout this chapter are obtained by splitting the datasets from chapter 4 with a 70% : 15% : 15% split, where the largest fraction of 70% goes to the training set. Each sub-dataset is then weighted according to the definitions in section 5.1. For the final evaluation of the performance of the GNNs, the validation set of the simulation group dataset was used such that the GNNs are evaluated on run-by-run MC data. The training set of the merged dataset defined in section 4.4 is obtained by concatenating and shuffling the training datasets of the simulation group and flat datasets. The analogous procedure is done for the validation and test set of the merged dataset, which ensures that the GNNs never saw the events in the validation and test set during training. Each training in this section is done with the same GNN in order to be able to compare the suitability of each dataset and reweighting method. Figure 5.7 shows the GNN used in this section, which is inspired by the ParticleNet architecture shown in fig. 3.7 with the difference that the model in fig. 5.7 uses 30 nearest-neighbors instead of 16 and that no dropout is used, where the latter decision was also made in [18, 19]. The complete set of hyperparameters of the GNN can be seen in appendix A.2.

5.2.1. Training on flat weighted merged dataset

As discussed in sections 4.1 and 5.1.2, the subtle differences in the neutrino event topologies shown in fig. 2.4 were suspected at the beginning of this thesis to be suitable features that the GNNs could exploit for tau identification. Hence, a training on the merged dataset with intermediate weighting was carried out, where $\alpha = 0$ and $\beta = 0.5$, which is equivalent to flat weighting as defined in section 5.1.2. Figure 5.8 shows the monitoring of the training, where the left plot shows the loss on the training data and the validation data, while the right plot shows the PR-AUC on the validation data. The PR-AUC evaluated on the training data is not shown in fig. 5.8 because of an issue with the calculation of this metric in OrcaNet. This issue will be resolved in a future version of OrcaNet, and it was checked that the calculation of the PR-AUC on the validation set

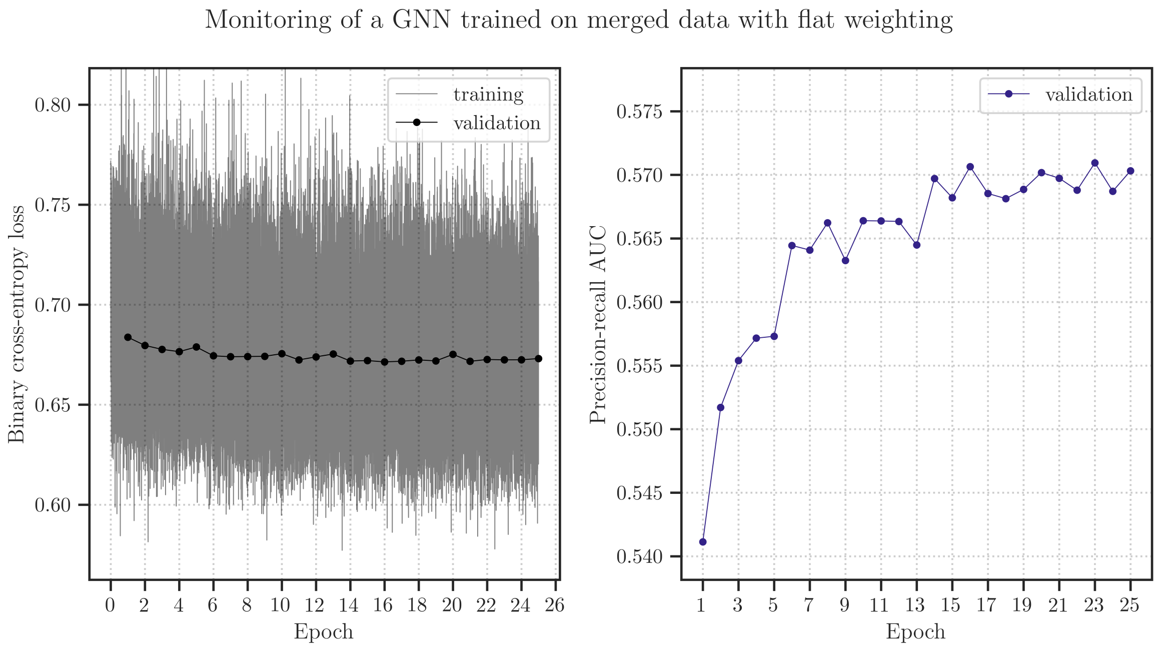


Figure 5.8.: The monitoring of the loss and the precision-recall AUC is shown for a training on merged data with flat weighting.

is not impaired by this issue.

The left plot in fig. 5.8 shows that the loss function decreases very slowly, which indicates that the GNN has trouble learning the discrimination between tau and nontau events. The right plot in fig. 5.8 looks like a steep increase in the PR-AUC at first glance, but if one pays attention to the scale of the y-axis, it can be seen that the metric only improves by about 5.6% since the completion of the first epoch. The model does not show signs of overfitting yet, i.e., no significant deviation exists between the training and validation loss. There is also no significant decrease in the PR-AUC on the validation data, which usually means that the GNN should have been trained further. However, compared to other models trained in this section, the model's performance is sufficiently bad to conclude that additional computation time would be a lousy investment of the limited GPU time budget. As shown in the right plot in fig. 5.8, the model performed best on the validation set after 23 epochs.

Hence, this model was further evaluated on the Honda weighted validation set of the simulation group dataset, resulting in the precision-recall curve shown in fig. 5.9. At small recall values, only a tiny fraction of the tau events is labeled positive by the GNN,

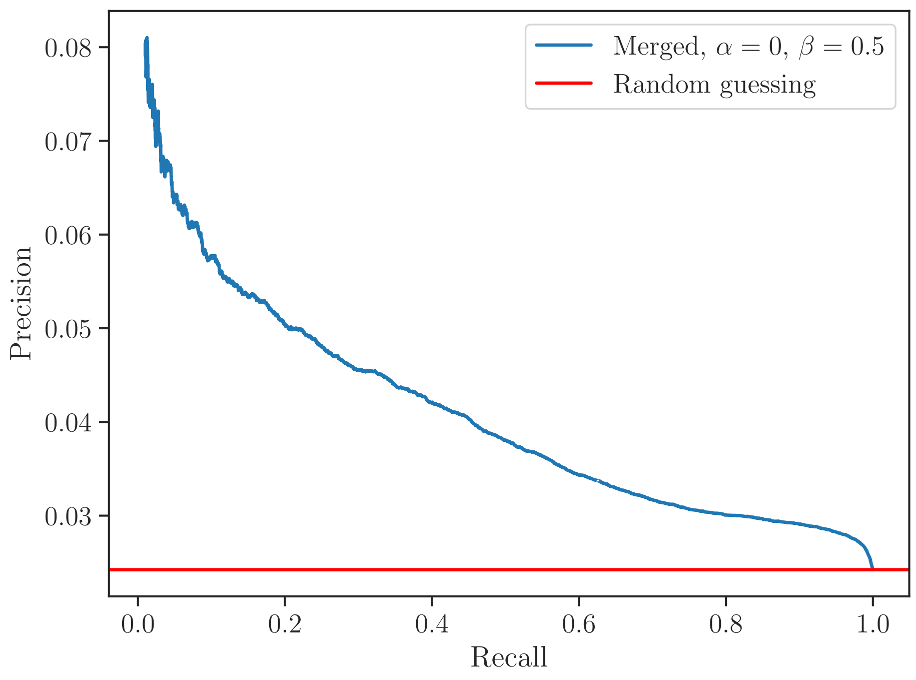


Figure 5.9.: The precision-recall curve for the GNN trained on the merged dataset with flat weighting, which is equivalent to intermediate weighting with $\alpha = 0$ and $\beta = 0.5$. The evaluation on the validation set was done with the weights of the 23rd epoch.

which causes these points in the curves to be susceptible to random fluctuations. This effect can be seen directly in fig. 5.9, where the curves become noisier for smaller recalls. To reduce the noise in the plots, all precision-recall curves throughout this thesis show only recall values above 1%. The curve shows that the GNN selects tau events better than a randomly guessing classifier, but the precision stays below 5% for recalls larger than 20%. The reason for this lack in performance was discussed in section 2.3, where it was explained that the event topologies of tau and nontau events look very similar in the detector. Since GNNs are a new approach to the problem of tau identification, it was worth trying out whether they can extract information solely from the slight differences in the event topologies. However, the result in this section, combined with the results in the next section, shows that focusing on the differences in the event topologies is still not feasible to tackle tau identification.

5.2.2. Intermediate weighting with varying α

This section tackles the question of which of the three datasets introduced in chapter 4 is the most suitable for tau identification and how much physical information from the Honda weighting should be supplied in the training set. To answer these questions, each of the three datasets (simulation group, flat, and merged) are weighted with 14 different values for the parameter α, resulting in a total of 42 weighted datasets. To determine the suitability of each of these weighted datasets for tau identification, one GNN with hyperparameters as shown in fig. 5.7 was trained for each of the weighted datasets. The trainings lasted at least 50 epochs for GNNs trained on the simulation group and the flat dataset, while trainings on the merged dataset only lasted for at least 25 epochs. The GNNs on the merged dataset trained for fewer epochs because the dataset is larger than the other two datasets, which leads to an increased training time. Trainings on the merged dataset needed about 160 min per epoch on an A100 GPU, resulting in a training time of minimum 2.78 d for GNNs on the merged dataset. On the other hand, trainings on the simulation group dataset lasted only for 73 min per epoch, and trainings on the flat dataset needed 81 min, which allowed them to train for more epochs in the same time. For each trained GNN, the weights from the epoch they performed best on their respective validation set are validated on the Honda weighted validation set of the simulation group dataset.

The results of the trainings can be seen in fig. 5.10, which shows the performance of the GNNs on the Honda weighted validation set dependent on the training dataset and the parameter α used during the intermediate weighting. It can be seen that the curve corresponding to the merged training set achieves the best PR-AUCs for all values of α, indicating that this dataset should be favored throughout the trainings in the following sections. Furthermore, it can be seen that the performance of the GNNs increases for increasing values of α, which demonstrates that physical information about the atmospheric neutrino flux can be utilized well by the GNNs, leading to enhanced performance. On the other hand, the performance on all of the three datasets is minimal when they are weighted flatly, supporting the conclusion drawn at the end of section 5.2.1 that flat weighting is not suitable for tau identification. The merged and the simulation group datasets have their maximum at $\alpha = 1$, while the flat dataset has its maximum at $\alpha = 0.9$. The limited computing time budget in this step of the thesis did not allow to train many GNNs per dataset and per α, which makes the curves in fig. 5.10 susceptible to random fluctuations. Hence, it can not be confidently stated from the

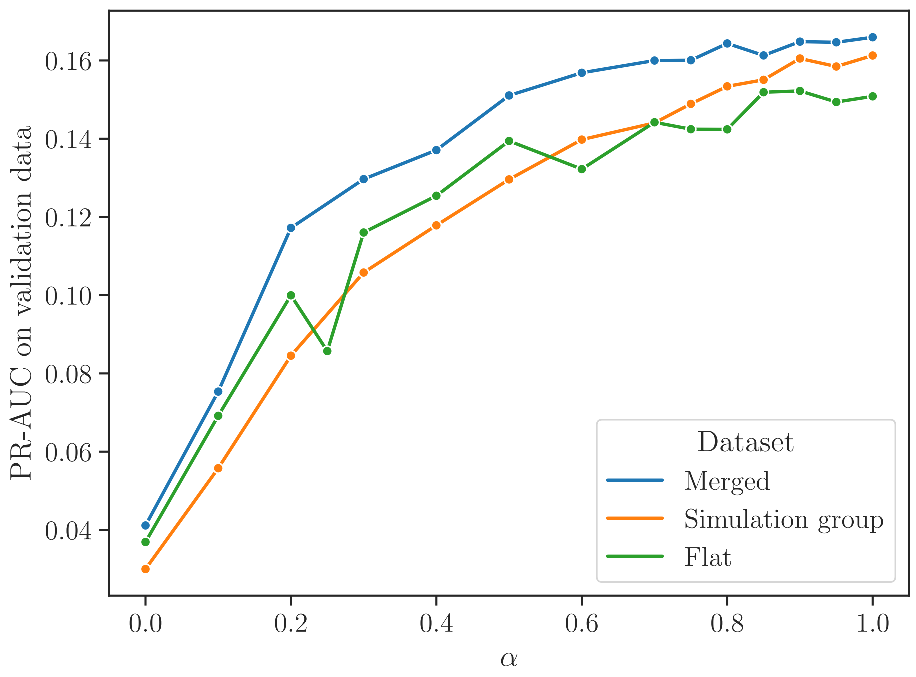


Figure 5.10.: The performance of the GNNs trained on differently weighted datasets is shown. While α is varied, the parameter β is fixed to 0.5 to obtain class balance.

available information which α is optimal. Since the merged dataset and the simulation group dataset had their maximum at $\alpha = 1$, this value is a probable candidate for a close-to-optimal value of α, which will be used in the following sections.

5.2.3. Intermediate weighting with varying β

The parameter β introduced in the discussion of intermediate weighting in section 5.1.3 determines the fraction of tau events in the dataset. Increasing this parameter scales the weights of all tau events up and all nontau events down so that the number of events remains conserved. This rescaling is helpful to counter the effects resulting from imbalanced classes discussed in section 4.1, which can be achieved by choosing $\beta = 0.5$ to obtain class balance. This section tests whether balanced classes are suitable for tau identification by choosing six different values for β for each of the three datasets, resulting in 18 weighted datasets. Analogous to the last section, one GNN with the hyperparameters shown in fig. 5.7 and in appendix A.2 is trained on each weighted

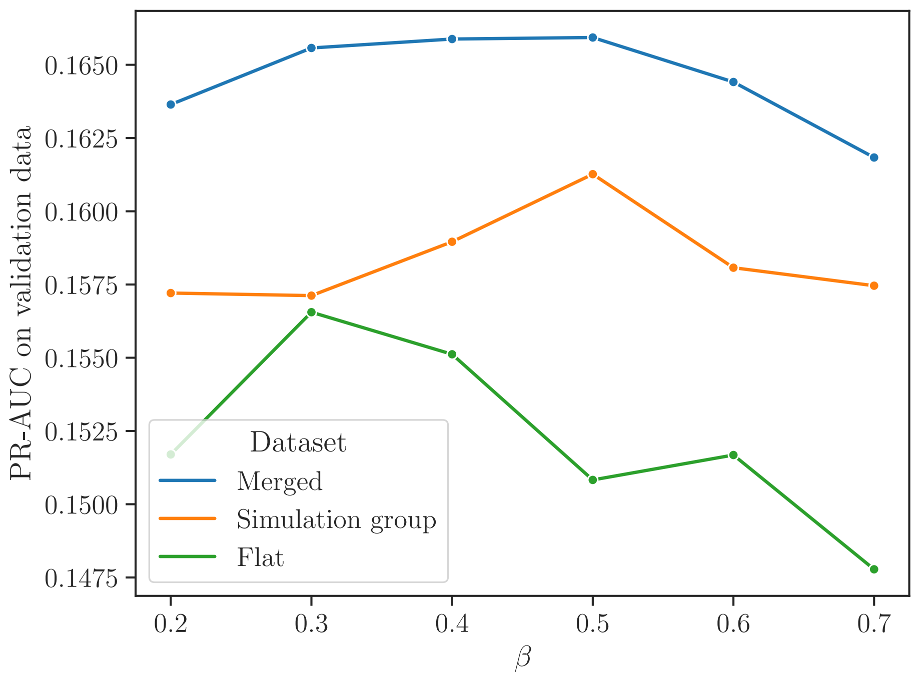


Figure 5.11.: The performance of the GNNs trained on differently weighted datasets is shown. The parameter α is fixed to 1, while the parameter β is varied.

dataset.

The results of these variations in β can be seen in fig. 5.11. As in the last section, the merged dataset performs best for all values of β. For both the merged dataset and the simulation group dataset, $\beta = 0.5$ resulted in the highest performance, while the flat dataset seems to favor a value of $\beta = 0.3$. The parameters $\alpha = 1$ and $\beta = 0.3$ on the flat dataset lead to a GNN that outperforms the current best GNN on the flat dataset from the last section, where the chosen values for the parameters were $\alpha = 0.9$ and $\beta = 0.5$. This result indicates further that the value $\alpha = 1$ could be a close-to-optimal candidate for tau identification. Analogous to the last section, the curves in fig. 5.11 are susceptible to random fluctuations since only one GNN could be trained per weighted dataset due to a limited GPU time budget. Since $\beta = 0.5$ resulted in the best performance for the merged and the simulation group dataset, it is assumed to be a probable candidate for a close-to-optimal value of β in combination with the value $\alpha = 1$, supporting the initial suspicion that balanced classes are a desirable property of a training set.

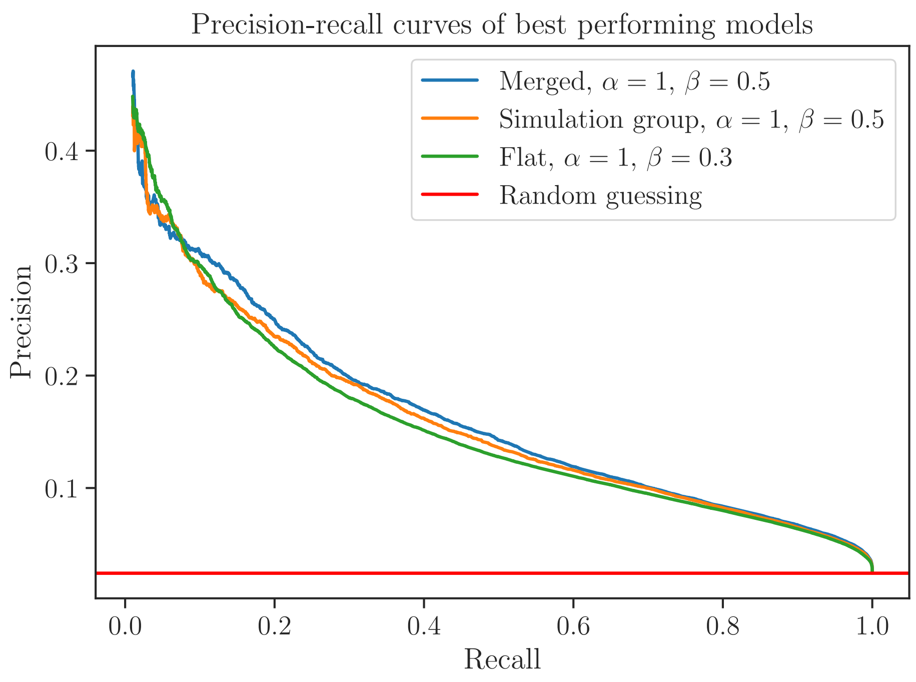


Figure 5.12.: The precision-recall curves of the best-performing model on each dataset is shown.

5.2.4. Evaluating the best performing models

Figure 5.12 shows the precision-recall curves of the best-performing models on each dataset found during the dataset optimization in the last two sections. The GNN trained on the merged dataset achieves better precisions than the other GNNs for recalls above 10%. The situation differs for smaller recall values, where the orange and green curves are often higher than the blue curve. As discussed in section 5.2, this recall regime is susceptible to random fluctuations.

The blue precision-recall curve for the training on the merged datasets shows a precision of 19.8% at a recall value of 30%, which is much better than the precision-recall curve achieved with the flatly weighted merged dataset in fig. 5.9, where the precision was only about 4.6% for a recall of 30%. While the PR-AUC of the GNN trained on flatly weighted merged dataset only reached 0.041, the GNN trained on the same dataset with intermediate weighting reaches a PR-AUC of 0.166 with the parameters $\alpha = 1$ and $\beta = 0.5$. The orange curve in fig. 5.12, which corresponds to the training on the simulation group dataset, has a PR-AUC of 0.161 and reaches a precision of 19.3% for a recall of 30%. Finally, the green curve corresponding to the training on the flat dataset reaches a PR-AUC of 0.157 with a precision of 18.0%for a recall of 30%.

The results of this dataset optimization give first hints that the production of additional Monte Carlo events in chapter 4 improved the performance of the GNNs, probably due to the increased statistics of tau events in the dataset. Further evidence for this suspicion will be collected and analyzed in section 6.4.2, which will result in a much more significant confirmation of the statement. Additionally, access to information about the atmospheric flux drastically enhances the performance of the GNNs compared to the extreme case of flat weighting. For the merged dataset, full Honda weighting ($\alpha = 1$) and balanced classes ($\beta = 0.5$) are probable close-to-optimal choices for the training dataset. Hence, this weighted training set is used for further trainings.

5.3. Manual hyperparameter optimization

The last section used the same GNN for all studies concerning optimizing the training dataset. In contrast, the weighted training set found in the last section is used now for training several GNNs with a different hyperparameter configuration. These hyperparameter configurations are sampled from the "hyperparameter space" $\mathcal{H}$, which is the set of all allowed hyperparameter configurations. Each hyperparameter configuration defines the values of all hyperparameters in a network, which makes it possible to build one concrete GNN from each hyperparameter configuration.

In this section, only four hyperparameters are varied, namely the batch size b, the number of nearest neighbors k used to connect the nodes in the graph, the number of EdgeConv layers e, and the number of dense layers d stacked after the global average pooling and before the output layer. EdgeConv blocks are added to the GNN in the same way as ParticleNet is doing it, i.e., by doubling the number of nodes in each layer of the kernel network. In contrast, dense layers are added to the network by halving their number of nodes with respect to the previous dense layer. The other hyperparameters are still the same as in the last section, i.e., they can be seen in appendix A.2. These fixed hyperparameters will not be discussed further in this section, and the hyperparameter space $\mathcal{H}$ can be entirely defined by listing the different combinations of tuples (b, k, e, d). The hyperparameter space chosen for this section can be described as a union of three subspaces, i.e.,

$$\mathcal{H} = \mathcal{H}_{\mathrm{S}} \cup \mathcal{H}_{\mathrm{M}} \cup \mathcal{H}_{\mathrm{L}}, \tag{5.17}$$

where the indices of the subspaces stand for small, medium, and large. The indices refer to the number of layers of the GNNs, where the small GNNs only have three EdgeConv blocks, the medium GNNs have four, and the large GNNs have five. The three subspaces are defined as

$$\mathcal{H}_{\mathrm{S}} = \{16\} \times \{30\} \times \{3\} \times \{1, 2\} \tag{5.18}$$

$$\mathcal{H}_{\mathrm{M}} = \{16, 32\} \times \{20, 30\} \times \{4\} \times \{1, 2\} \tag{5.19}$$

$$\mathcal{H}_{\mathrm{L}} = \{16, 32\} \times \{20, 30\} \times \{5\} \times \{1, 2, 3\}, \tag{5.20}$$

which results in a total of 22 different hyperparameter configurations after performing the cartesian products. This number of hyperparameter configurations was chosen because a maximum of 24 GPUs could be used in parallel for these studies. The two remaining GPUs not used for the manual hyperparameter optimization were used as a backup for other trainings that ran in parallel. The three subspaces defined above were chosen with the goal of testing models with more free parameters than the GNN used in the last section, which had 365 903 trainable parameters. For comparison, the largest GNNs in this manual hyperparameter optimization, i.e., the GNNs with $(e, d) = (5, 3)$, have 6 605 647 trainable parameters. The batch size $b = 32$ and the nearest neighbors $k = 20$ were added to reduce the training time needed by the larger GNNs since a larger batch size results in fewer time-expensive weight updates per epoch, and a smaller number of nearest neighbors reduces the time spent on finding the nearest neighbors of each node in the graph. The GNNs were trained on A40 GPUs until they either reached 50 epochs or until a computation time limit of about two weeks was reached, where the latter was necessary to terminate the training of the larger GNNs.

A detailed table with the results from the manual hyperparameter optimization is added to the appendix in table A.1. The table shows that the best GNN achieved a PR-AUC of 0.174, improving over the best PR-AUC from the last section, which had the value 0.166. Both the second and third places also reached a PR-AUC of about 0.174, which shows that the performance of the best-performing GNNs is very similar. The differences can only be seen at decimal places that are further trailing, which are probably not statistically significant. The three best-performing models' precision-recall curves are shown in fig. 5.13. In addition, the precision-recall curve of the best-performing model of

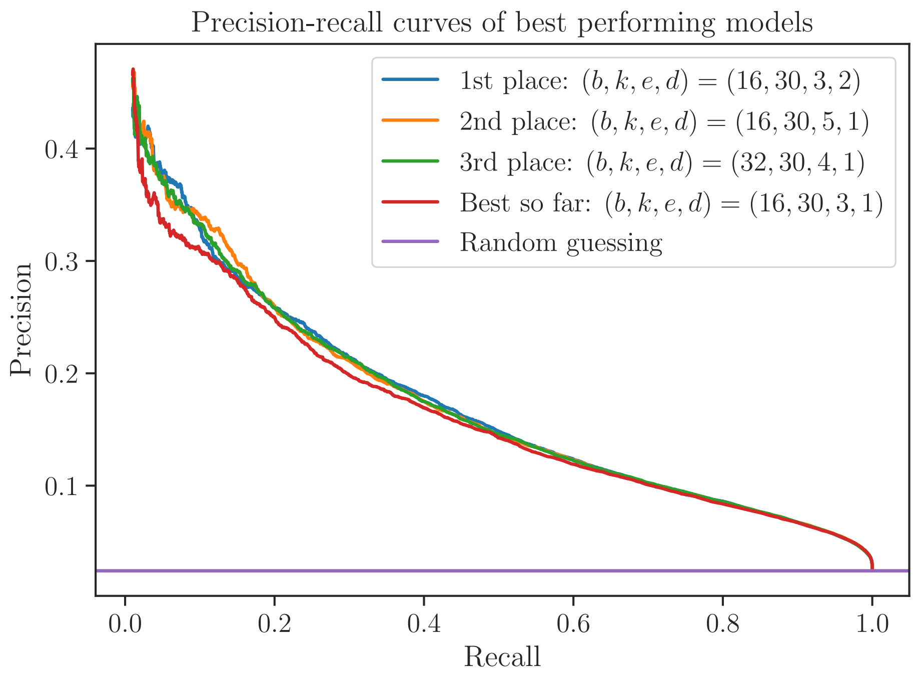


Figure 5.13.: The precision-recall curves of the three best-performing GNNs is shown. Additionally, the precision-recall curve from the best-performing GNN of the last section is shown in red for comparison.

the last section is also plotted, which is considerably worse than the new best-performing models. While the best-performing model from the last section reached a precision of 19.8% for a recall of 30%, the best-performing model in this section reaches a precision of 21.4% for the same recall value. The second and third model reach precisions of 21.1% and 21.4%, showing that all three models perform very similar at the chosen recall value and that the best-performing models does not necessarily have to dominate the other models at all recall values. The model in the first place has the highest precisions for recalls above 20%. For lower recall values, the second place can reach higher precisions than the first for some recall values, but as discussed before, this recall range is susceptible to random fluctuations.

The limited statistics of this manual hyperparameter optimization make it impossible to reach conclusions on the best hyperparameter values for this GNN architecture. Hence, the problem of hyperparameter optimization is tackled further in the following chapter, where an automatic hyperparameter optimization is performed to collect much more data about several hyperparameters of the GNN architecture in an automated way.

6. Automatic hyperparameter optimization

6.1. Motivation

An automatic hyperparameter optimization (AHPO) is a procedure in which hyperparameter configurations are repeatedly and automatically sampled from the hyperparameter space $\mathcal{H}$ and subsequently trained and evaluated. The sampling and evaluations of the hyperparameter configurations are managed by various algorithms optimized for using the limited computing time budget as efficiently as possible to find the best hyperparameter configuration. Automatic hyperparameter optimizations are a modern approach to tackle hyperparameter optimization. The traditional method of manually optimizing the hyperparameters, which was performed in section 5.3, consists of a trial-and-error based approach in which a (typically) small number of hyperparameter configurations is chosen and evaluated based on subjective prior experience of the machine learning developer. This process can become inefficient if there is no reliable prior experience because the machine learning developer is confronted, e.g., with a new model architecture or with a new task on which neither personal experience nor experience in the machine learning community has been collected so far. Additionally, manually optimizing hyperparameters becomes often intransparent and non-reproducible because many academic papers only state the final best model found during the hyperparameter optimization, preventing the community from profiting from the experiences collected during the optimization. The latter often happens because it is lengthy to describe the entire hyperparameter space that was manually chosen. The thorough description of the manual hyperparameter space in section 5.3 was possible because the optimization restrained itself to varying only four categorical, i.e., non-continuous, hyperparameters with values chosen in a grid-like structure. A thorough and reproducible description of a manual hyperparameter optimization is often not feasible if many hyperparameters are varied in a non-grid-like structure.

All of these problems are tackled by automatic hyperparameter optimizations (AHPOs). After the hyperparameter space $\mathcal{H}$ was defined by the machine learning developer, the variety of algorithms implemented to search through the hyperparameter space allows an efficient optimization even on problems where no prior experience was collected. Furthermore, an AHPO can be compactly described in a transparent and reproducible manner by specifying the chosen hyperparameter space and the choice of the particular algorithms used to define the optimization process. During an AHPO, hundreds to thousands of models are typically trained and evaluated, making it difficult to track all the results. Hence, many frameworks for AHPO additionally offer functionality that takes care of the bookkeeping and provides easy access to the data collected during the AHPO, which can subsequently be used to analyze the results. This data can be used to study which hyperparameters have the most significant impact on the performance of the models and to find suitable values for these hyperparameters, which could be beneficial for both the task at hand as well as for similar future tasks to which these results may be extrapolatable.

Hence, the decision was made that the final optimization step in this thesis should consist of an automatic hyperparameter optimization, which is the first optimization of that kind performed in KM3NeT. The three main goals of the optimization are the further improvement of the performance of the GNNs, the exploration of GNN architectures that deviate from the ParticleNet architecture shown in fig. 3.7, and the identification of the hyperparameters that have the most significant impact on the models' performance. Since an AHPO of this scale needs many GPU-hours of computing time, an application for a compute time project was submitted to the "Erlangen National High Performance Computing Center" (NHR@FAU), which was approved after a review of its technical feasibility as a continuation of a DFG-project that was scientifically reviewed in the past. The project is listed under the acronym "AHPO4TauID" with the project ID "b173dc". In total, 46 000 GPU-h on A100 GPUs were granted on the Alex cluster.

6.2. Ray Tune

There exist many frameworks that offer functionality to define and execute an AHPO efficiently, e.g., KerasTuner[43], Optuna[44], scikit-learn[45], and Ray Tune[46], where the latter was used in this work. Ray Tune is a library for automatic hyperparameter optimizations with various pre-implemented state-of-the-art algorithms. Ray Tune is part of the Ray framework[47], which offers functionality and libraries for easy scaling of machine learning tasks from running on a local machine to a cluster of multiple computing nodes. Ray provides extensive documentation that shows, e.g., how to run it on a computing cluster like Alex using the workload management system "Slurm"[35]. This extensive documentation and the variety of pre-implemented algorithms were the main reasons why Ray Tune was chosen over other AHPO frameworks.

The central object that has to be defined by the machine learning developer to perform an AHPO with Ray Tune is an instance of the `ray.tune.Tuner` class. The hyperparameter optimization can be started after a proper configuration by calling the method `Tuner.fit()`. The `Tuner` class accepts many different parameters as input. A subset of these parameters will be discussed in this paragraph to demonstrate how Ray Tune works without going too far into the technical details.

The first building block is the hyperparameter space, represented as a dictionary mapping some arbitrary user-specified names of hyperparameters to a function specifying how to sample this hyperparameter from a random distribution. An example would be the dictionary entry `"learning_rate" : tune.loguniform(1e-3, 1e-2)`, which specifies that the learning rate should be sampled uniformly in log space for values between 1×10^{-3} and 1×10^{-2}. The hyperparameter space determines all hyperparameters that should be varied during the optimization, in which ranges they should be varied, and from which random distribution they should be sampled.

The second building block that has to be specified to the `Tuner` is the `trainable`. The `trainable` can be defined as a function or class. In this thesis, the definition as a function is sufficient. The `trainable` receives a hyperparameter configuration as input, i.e., the result of one sampling from the hyperparameter space. The `trainable` is responsible for building the `model` with the hyperparameters specified in the hyperparameter configuration. Additionally, it is responsible for starting the training of the `model`, which is done with the `model.fit()` in the case of using Keras. Due to this very general way of defining a model and starting a training, Ray Tune is compatible with most machine learning frameworks, which made it possible to use the functionality provided by OrcaNet[28] to build the models and to preprocess the data for the training. The `trainable` is also responsible for storing the weights after each epoch and reloading the weights from the last checkpoint if a training was interrupted. Since the Slurm system on the Alex cluster does not allow jobs exceeding a runtime of 24 h, implementing the latter functionality was crucial. Ray offers an interface making storage and reloading of weights on distributed systems very easy. The class `ray.air.integrations.keras.ReportCheckpointCallback` is offered to store weights, which creates a `ray.train.tensorflow.TensorflowCheckpoint` after each epoch. The last checkpoint of a training can easily be loaded by using the function `ray.air.session.get_checkpoint()`, which returns the last `TensorflowCheckpoint`. This checkpoint can be used to obtain the model weights with a call to its method `TensorflowCheckpoint.get_model()`. Hence, machine learning developers do not have to interact manually with the filesystem to store the model weights, which drastically simplifies the scaling to distributed systems.

The third building block is the `tune.TuneConfig`. It specifies, e.g., which metric should be used by the AHPO to compare the performance of different models. In the case of this thesis, the PR-AUC achieved on the Honda weighted validation set of the simulation group dataset was used as a performance metric. The `TuneConfig` also defines which algorithms should be used during the AHPO. Ray Tune classifies its algorithms into two classes, which are called “search algorithms” and “schedulers”. Search algorithms are responsible for suggesting hyperparameter configurations. Simple search algorithms exist, like grid search, which suggests hyperparameter configurations sampled from a grid (used in section 5.3), and random search, which randomly samples from the hyperparameter space. Additionally, some more sophisticated algorithms like `BayesOptSearch`, which uses previous results in a Bayesian optimization to suggest promising hyperparameter configurations in the future, are implemented. In contrast, schedulers are responsible for scheduling the trainings efficiently, i.e., assigning the limited computing time as efficiently as possible. They can achieve this, e.g., by terminating trainings early that show unpromising performance, which frees resources that can be used to start the training of the next sample from the hyperparameter space. Examples of schedulers are `HyperBand` and `ASHA`. The Ray Tune FAQ offers advice on which combination of search algorithm and scheduler to choose in which situation[48]. The “go-to solution” for smaller problems is described as using a random search combined with the `ASHA` scheduler. A rough estimation of the number of models that can be evaluated with 46 000 GPU-hours predicted 1000-2000 different hyperparameter configurations, which was assumed to fall into the category of a “small problem”. Hence, the advice of the FAQ was followed, and the random search algorithm in combination with the `ASHA` scheduler was selected. The latter will be discussed in section 6.3.

The fourth building block is the `air.RunConfig`, which is a rather technical object determining where the results of the AHPO should be stored and how different computing nodes should be synchronized with each other. Again, Ray offers various abstractions for these options, making scaling to distributed systems easy. However, a detailed discussion of these abstractions would be too technical for this thesis. The details of the AHPOs performed in this chapter can be found in [49]. It should be added for readers interested in performing their own AHPO with Ray Tune that a short documentation of the experiences with the framework will be written for the NHR@FAU a few months after submission of this thesis, which will include the most common pitfalls that new user could stumble upon. This short documentation will also briefly cover how to set up Ray on the Alex cluster, which should be readily generalizable to any cluster using Slurm.

6.3. The Asynchronous Successive Halving Algorithm

The Asynchronous Successive Halving Algorithm (ASHA) is a scheduling algorithm proposed by [50]. It is an extension of the Successive Halving Algorithm (SHA), which was proposed by [51] for usage in hyperparameter optimization. In SHA, one starts with n hyperparameter configurations, which will be referred to in the following as “configurations” for brevity, and a minimum resource r, which could be, e.g., computing time or the number of trained epochs. In the first step, the resource r is assigned equally to the n configurations, which are then trained until they completely used their assigned resource. After all of the configurations are finished consuming their resources, the top $1/\eta$ fraction of all configurations are identified and allowed to be trained further, where η is the “reduction factor”. The other configurations are eliminated. In the original paper proposing SHA for hyperparameter optimization, $\eta = 2$ was chosen, which results in the worst half of configurations being eliminated, explaining the algorithm’s name. The remaining $n\eta^{-1}$ configurations are allowed to use the same total amount of resources as before, meaning that $r\eta$ resources are assigned to each configuration. These configurations are then trained until they consumed their resources, and the process repeats until only one configuration remains, which happens after $\lceil \log_\eta(n) \rceil$ steps. Each step is called a "rung" labeled by k. In each rung k, $n\eta^{-k}$ configurations are trained with $r\eta^k$ resources per configuration, leading to a total resource consumption of $B = nr\lceil \log_\eta(n) \rceil$.

The Hyperband algorithm is an extension of SHA that tackles the "n vs B/n" problem in SHA[52]. This problem refers to the circumstance that it is not known a priori how the number of configurations n, which is an input parameter to SHA, should be chosen optimally. Each of the n configurations in SHA receives, on average, B/n resources. If n is chosen to be large, many configurations are evaluated, but each receives only a small amount of resources B/n on average. Hence, in situations where the good configurations can only be differentiated from the bad ones after a large amount of resources was consumed, choosing a small number of configurations n which receive a large amount of resources B/n on average may be recommended. Hyperband introduces the concept of "brackets" labeled by s to solve this problem. As described in [52], each bracket s carries out one complete execution of SHA with a different number of configurations n, which means that the Hyperband algorithm performs a grid search over different values for n with a fixed total budget B. The concept of brackets is used in ASHA, where bracket s is defined to train $n\eta^{-s-k}$ configurations, to which $r\eta^{s+k}$ resources are assigned per configuration, in the kth rung. Hence, the zeroth bracket $s = 0$ corresponds to one complete execution of SHA with many configurations that receive only a few resources per configuration, meaning that the configurations are eliminated aggressively. Brackets with larger values of s start with fewer configurations, which are trained with more resources before an elimination round is performed.

ASHA is an extension of SHA and Hyperband that makes these algorithms run asynchronously[50]. In SHA, all configurations in the kth rung must use their assigned resources before the best $1/\eta$ of configurations can be identified and promoted to the next rung $k + 1$. If some configurations need more time to finish a training with their assigned resources, computing resources like GPUs remain unused. To counter these straggler issues, ASHA defines a completed configuration in the kth rung to be "promotable" if it places in the top $1/\eta$ fraction of completed configurations in the kth rung and if it has not already been promoted to the next rung yet. If one worker process is free, the promotable configuration currently in the furthest rung is assigned to the worker asynchronously, i.e., the worker does not wait for all the configurations of the rung to finish their training. This promotable configuration is therefore promoted to the next rung through this process. If there is no promotable configuration, the free worker gets assigned a new configuration starting in the 0th rung. This procedure ensures that the computing resources are always fully utilized.

The ASHA algorithm is implemented in Ray Tune via the `AsyncHyperBandScheduler` class in `ray.tune.schedulers`, which receives a few parameters that determine the algorithm's behavior. Five of these parameters are discussed in more detail in this paragraph. The first parameter is the `time_attr`, which can be any monotonically increasing quantity, e.g., the training time or the number of batches processed. This thesis uses the number of trained epochs as `time_attr`. The second parameter is the `max_t`, which defines the maximum amount of resources that can be assigned in total to a single configuration before its training is terminated. This parameter is used in this study to specify the maximum number of epochs a configuration can train. The `grace_period` is a parameter equivalent to the minimal resource r used in the description of the ASHA algorithm above. It determines the minimum number of epochs given to a configuration before the first round of elimination occurs. The next relevant parameter is the `reduction factor`, which is equivalent to the parameter η defined above. Finally, the number of brackets has to be defined with the `brackets` parameter, which fixes the range of the parameter s defined above to $s \in \{0, \ldots, \text{brackets} - 1\}$. Each configuration is assigned randomly to one of the brackets, where a softmax function gives the probability of getting assigned to a bracket. This softmax function ensures that brackets with smaller s, i.e., brackets that terminate configurations more aggressively, should have an increased probability of getting a configuration assigned, which prevents clustering of the configurations in the less aggressive brackets.

6.4. Performing automatic hyperparameter optimizations

The 46 000 GPU-hours granted to this project were spent on four automatic hyperparameter optimizations, which differed in the structure of their network architectures, their defined hyperparameter spaces, their different combinations of parameters configuring the `AsyncHyperBandScheduler`, and in one case even in the training set that was used. The four hyperparameter optimizations can be grouped into two categories, where the main differences between the two categories lie in the type of GNN architecture that was tested.

6.4.1. Generalized architectures

Defining the hyperparameter space

In contrast to the ParticleNet-like architectures discussed later, the generalized architectures refer to a set of architectures not constrained to specific properties of the ParticleNet architecture discussed in section 3.5.3. The ParticleNet architecture consists of three EdgeConv blocks. Each of these blocks has a kernel network with exactly three layers, which will be called "kernel layers" from now on. Each kernel layer in an EdgeConv block has the same number of neurons, starting with the first block having 64 neurons. The number of neurons per kernel layer is doubled after each EdgeConv block until a peak width of 256 neurons is reached in the third block. The output of the EdgeConv block with peak width is pooled globally and fed into one dense layer with the same number of neutrons as the kernel layers of the last EdgeConv block. The output of the dense layer is then reduced to one neuron in the output layer. These "rules" to construct a ParticleNet-like GNN were followed during the manual hyperparameter optimization in section 5.3. The generalized architectures used in this category do not have these constraints. The goal of this category of AHPOs is to explore GNN architectures that are deviating significantly from the ParticleNet-like architectures, with the hope of finding new rules to construct GNNs that achieve improved performance compared to the ParticleNet-like architectures in the task of tau identification.

The generalized architectures in this category can be structured into two "build methods". The first build method consists of GNNs doubling their number of neurons in the kernel layer after each EdgeConv block, which is what ParticleNet-like architectures do. This build method results in an exponential increase of the number of neurons in the kernel layers by a factor of two after each EdgeConv block, which is why this build method will be referred to as "PerBlock" from now on. The other build method is referred to as "PerKernelLayer" because the number of neurons exponentially increases after each kernel layer, resulting in a GNN with consistently increasing kernel layers until reaching the GNN's peak width. An example of a PerKernelLayer GNN can be seen in fig. 6.1.

The PerBlock and the PerKernelLayer architectures share the same hyperparameters, with one exception that will be discussed later. The different hyperparameters consist of the learning rate, the batch size, the option to use shortcut connections in the EdgeConv blocks, the option to use batch normalizations in the network, the option to use dropout in

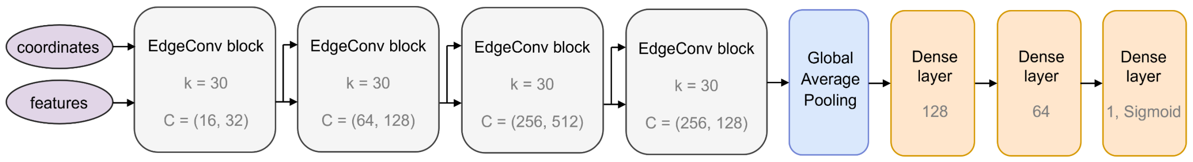


Figure 6.1.: A GNN built with the PerKernelLayer method. Note the exponential increase of neurons in each kernel layer, the exponential decrease of the EdgeConv block after the peak, and the exponentially decreasing dense layer.

the dense layers and the choice of the particular dropout rate, and the number of nearest neighbors to which each node in the graph gets connected. Additionally, there exists a hyperparameter that determines the number of neurons in the first kernel layer of the first EdgeConv block, from which the number of neurons in all subsequent kernel layers can be determined by the rules that were defined above for the build methods PerBlock and PerKernelLayer. The number of kernel layers per EdgeConv block is defined by a hyperparameter which has a value of three in the case of ParticleNet (fig. 3.7) and a value of two in the example of the GNN in fig. 6.1. Two hyperparameters determine the number of EdgeConv blocks. The first parameter determines the number of EdgeConv blocks before the peak, i.e., the number of EdgeConv blocks with an exponentially increasing number of neurons determined in detail by the build method PerBlock or PerKernelLayer. The second parameter determines the number of EdgeConv blocks after the peak, which has an exponentially decreasing number of neurons in their kernel layers. This parameter has a value of zero in the case of ParticleNet and a value of one for the GNN shown in fig. 6.1. The last hyperparameter is the number of dense layers before the output layer, where the first dense layer has the same number of neurons as the last kernel layer of the last EdgeConv block. Subsequent dense layers have an exponentially decreasing number of nodes.

The complete list of all hyperparameters, their chosen parameter ranges, and the values of the parameters configuring the `AsyncHyperBandScheduler` is added to the appendix. In total, two AHPOs were performed for the generalized architectures. The parameters chosen for the first AHPO are shown in listing A.1. It can be seen that the first AHPO uses a `grace_period` of one epoch in combination with only one bracket, which results in a very early and aggressive termination of bad trials. Hence, the first AHPO will be called "GerneralizedAggressive" due to its two main characteristics, i.e., using generalized GNN architectures combined with aggressive termination. The parameters defining the second

AHPO performed with generalized architectures can be seen in listing A.2. This AHPO uses a `grace_period` of four combined with three brackets, which is a much more relaxed termination behavior compared to the behavior in the GeneralizedAggressive AHPO, which is the reason why the second AHPO will be referred to as GeneralizedRelaxed from now on. The first round of eliminations in the GeneralizedAggressive AHPO takes place after the first epoch, while the GeneralizedRelaxed AHPO uses three brackets in which the first elimination round happens either after 4, 8, or 16 epochs, respectively. The reason why the GeneralizedRelaxed AHPO was performed is that the GeneralizedAggressive AHPO led to GNNs that performed worse than the models found in the manual hyperparameter optimization in section 5.3, which will be discussed in more detail in the next section. This result gave rise to the suspicion that the GeneralizedAggressive AHPO led to models that survived the early elimination rounds due to their ability to learn quickly in the first few epochs but failed to improve significantly in later epochs. Hence, the GeneralizedRelaxed was performed to allow the models to train a few epochs before getting evaluated in the first termination round.

The hyperparameter ranges of the GeneralizedAggressive and the GeneralizedRelaxed AHPO are very similar. They only differ slightly in the parameter ranges for the nearest neighbors, which was reduced for the GeneralizedRelaxed AHPO in order to accelerate the training of the models, and in the hyperparameter ranges determining the numbers of layers, the number of neurons in the first kernel layer, and the number of kernel layers per EdgeConv block. The hyperparameter ranges were manually selected, with inspiration drawn from [18, 19, 27] and from the experiences of the manual hyperparameter optimization in section 5.3. The hyperparameters affecting the number of free parameters in the model had to be chosen such that the resulting models were not too small or too large. Hence, to ensure that these hyperparameter ranges were chosen to result in models of moderate size, different hyperparameter ranges were studied by repeatedly sampling from different hyperparameter spaces, compiling the resulting models, and counting the number of free parameters. Figure 6.2 shows the distribution of free parameters resulting from sampling 2000 models for both the PerBlock and the PerKernelLayer build method using the hyperparameter ranges defined in listing A.2. The hyperparameter ranges of both the GeneralizedAggressive and the GeneralizedRelaxed AHPO were chosen such that the number of free parameters lies in the range $[3 \times 10^5, 9 \times 10^6]$. The lower bound of the range was chosen because it was suspected that architectures larger than the ParticleNet-like architecture used in section 5.2, which had 365 903 free parameters,

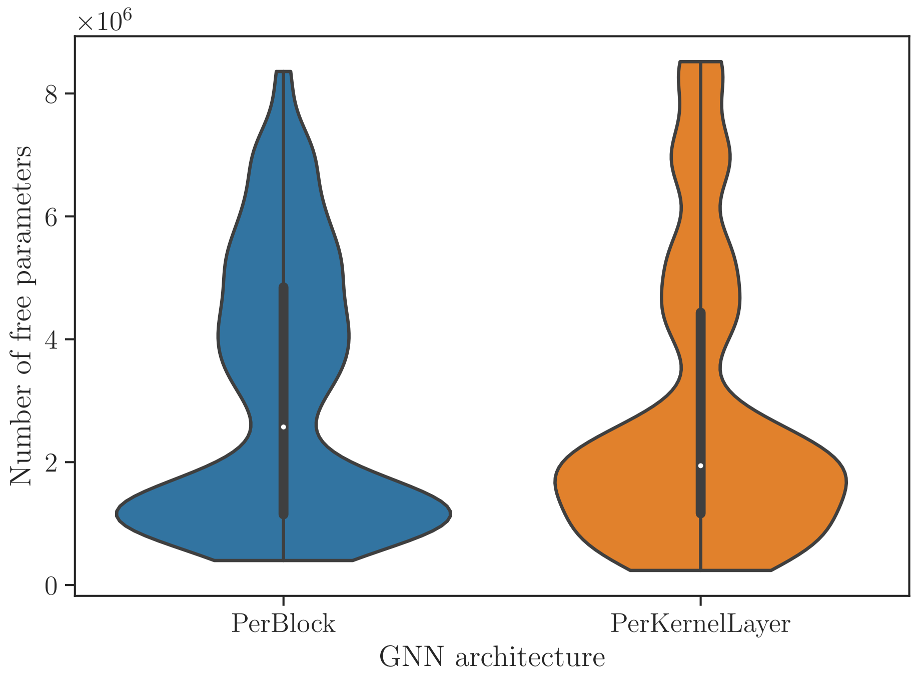


Figure 6.2.: A violin plot showing the distribution of free parameters in models sampled from the hyperparameter space chosen for the GeneralizedRelaxed AHPO. The width of the violin indicates how many models were sampled with the corresponding number of free parameters on the y-axis. The white point indicates the median, while the black bar around the median shows the interquartile range.

would lead to better performance. The upper bound was chosen such that the GNNs would finish one epoch of their training in less than 24 h, which was necessary since the Alex cluster does not allow jobs to run more than 24 h and the model's weights were only checkpointed after finishing an epoch. Figure 6.2 shows that sampling smaller models is more probable than sampling a larger model, which is a desired property since larger models need more training time per epoch. If smaller and larger models were drawn with equal probability, the shorter training time of the smaller models would cause them to reach the termination rounds quicker, which could lead to a replacement of the smaller model by a larger model. Hence, after some time, most of the models currently trained would be larger models, which is not desired.

As stated before, the GeneralizedAggressive and the GeneralizedRelaxed AHPO are described by the same hyperparameters with one exception. The exception is that the GeneralizedRelaxed AHPO additionally introduces the possibility of having an exponentially decreasing learning rate, which can be seen by the `use_lr_schedule` parameter in listing A.2. A decay in the learning rate results in smaller gradient update steps for later epochs, which can help the model to converge to a local minimum of the loss function. If a hyperparameter configuration with learning rate decay is sampled from the hyperparameter space, the learning rate of the model will decay exponentially with a decay rate chosen such that the learning rate will be reduced by a factor of ten every 20 epochs.

Analyzing the impact of different hyperparameters

In total, 567 GNNs were trained during the two AHPOs using generalized architectures. The GeneralizedAggressive AHPO trained 448 GNNs in about 4.5 d, while the GeneralizedRelaxed AHPO trained 119 GNNs in about 3 d. The trainings were performed on eight computing nodes housing eight A100 GPUs each, resulting in a parallel usage of 64 GPUs. All models in this section were trained on the training set of the merged dataset with intermediate weighting, where $\alpha = 1$ and $\beta = 0.5$. The validations were performed on the Honda weighted validation set of the simulation group dataset. The weights from the epoch with the highest performance on the validation data were chosen to evaluate a model's performance.

Figure 6.3 compares the performance distributions of the GeneralizedAggressive and the GeneralizedRelaxed AHPOs. The comparison is not entirely justified since the hyperparameter spaces of the two AHPOs and the parameters configuring the scheduler are different for both AHPOs. Since the GeneralizedAggressive AHPO terminates bad models already after the first epoch, the distribution would be biased to lower performances compared to the GeneralizedRelaxed AHPO since the models terminated after the first epoch have worse performances compared to the GeneralizedRelaxed AHPO where they would have been trained for more epochs. To counter this bias, fig. 6.3 shows only models that have their best epoch after the fourth epoch, which is the epoch where the GeneralizedRelaxed AHPO has its first termination round. Hence, fig. 6.3 answers which of the two AHPOs produced better models after their fourth epoch. In fig. 6.3, the GeneralizedAggressive AHPO produced better models than the GeneralizedRelaxed AHPO for both build methods. Since the confidence intervals of the medians indicated by the notches in the boxes of the boxplot are not overlapping, these

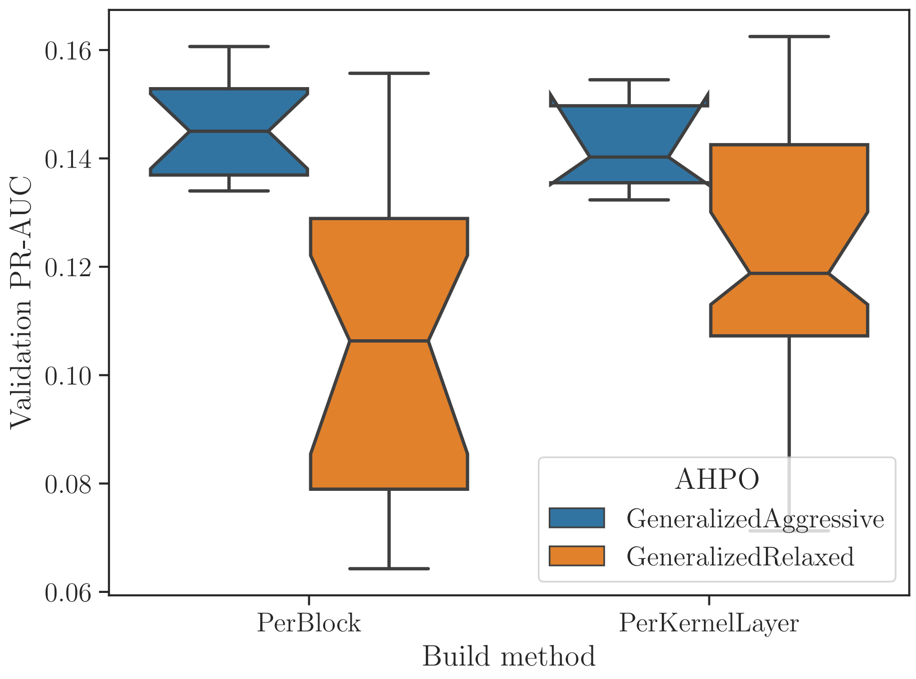


Figure 6.3.: Performance distributions of the models trained in the GeneralizedAggressive and the GeneralizedRelaxed AHPO for both build methods. The notches in the boxes represent 95% confidence intervals calculated by bootstrapping with 10 000 samples.

differences are considered significant. One could argue that the GeneralizedAggressive AHPO had more GPU-hours than the GeneralizedRelaxed AHPO, i.e., more time to find well optimized models. However, it is unlikely that the large difference between the medians in fig. 6.3 could be caught up by spending more computation time. It is more plausible that the aggressive termination behavior of the GeneralizedAggressive sorts out many bad-performing models very early, such that higher epochs are only assigned to models that performed well. Hence, a more aggressive approach than in the GeneralizedRelaxed AHPO is recommendable to find better-performing models in the limited computing time budget. Since the GeneralizedAggressive AHPO has more statistics and better performing models, the rest of this section focuses solely on this AHPO.

It has to be noted in the analysis of the results that distributions shown in boxplots like fig. 6.4 do not represent the performance distribution of fully trained GNNs, i.e., GNNs trained for many epochs until overfitting starts. Instead, the performance distribution of

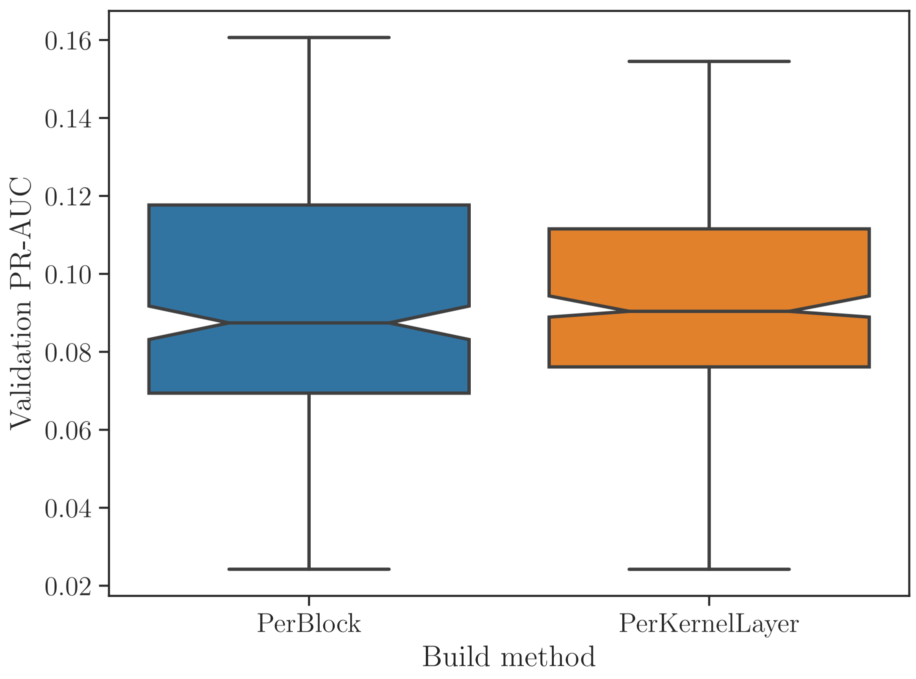


Figure 6.4.: Performance distributions of the two build methods in the GeneralizedAggressive AHPO.

GNNs is shown whose trainings were scheduled by the ASHA algorithm, i.e., including the performances of GNNs that were terminated early. Hence, if a boxplot shows that the performance distributions improve for a certain hyperparameter, this only shows that GNNs with that hyperparameter perform better during an AHPO with the ASHA scheduler. However, suppose one assumes that it is improbable that an early-terminated GNN will catch up to good-performing models if it is trained for more epochs. In that case, hyperparameter values superior during an AHPO with the ASHA scheduler will also be superior when fully trained GNNs are compared. The necessity for this assumption is a drawback of using the ASHA algorithm. However, without early termination, the problem becomes intractable since full trainings of thousands of GNNs are too time-expensive.

Figure 6.4 shows the performance distributions of the two build methods PerBlock and PerKernelLayer in the GeneralizedAggressive AHPO. The distributions for both build methods look very similar, with a slightly higher median for the PerKernelLayer build method. On the other hand, the distribution of the PerBlock method is wider than the PerKernelLayer distribution, which hints that the PerBlock method produces good-performing models more frequently, but this has to be stated carefully since the deviations

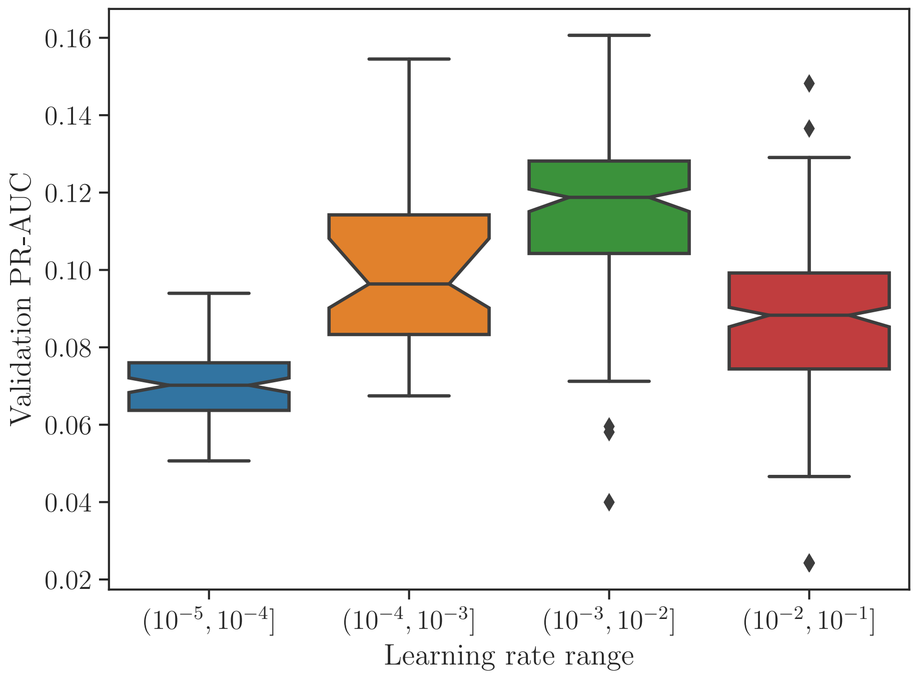


Figure 6.5.: The performance distributions for different orders of magnitude of the learning rate for the GeneralizedAggressive AHPO. The learning rates were sampled uniformly in logspace as defined in listing A.1.

are too small to be considered significant. Hence, it can be concluded that there are no hints in the data suggesting a significant improvement in using the PerKernelLayer method over the ParticleNet-like PerBlock method for obtaining very well-performing models.

The learning rate is often considered to be the hyperparameter with the most significant impact on the model's performance. This statement is checked in fig. 6.5, where the performance distributions for different orders of magnitude of the learning rate are shown. The figure shows a significant improvement of the models with learning rates in the range $[1 \times 10^{-3}, 1 \times 10^{-2}]$ over the other orders of magnitude, which confirms that the learning rate is a very significant hyperparameter. Hence, the range of the learning rate in the AHPOs shown in the next section will be fixed to $[1 \times 10^{-3}, 1 \times 10^{-2}]$ to prevent further sampling of hyperparameter configurations with an unsuitable learning rate.

Figure 6.4 shows that the models trained during the GeneralizedAggressive AHPO do not perform as well as the models trained in the manual hyperparameter optimization in section 5.3. Possible reasons for this are that the hyperparameter ranges defined in listing A.1 include too many possible hyperparameter configurations, making sampling of a model with good performance very improbable. An example for this argument is the learning rate, which was chosen to be uniformly sampled from a log space in the range $[1 \times 10^{-5}, 1 \times 10^{-1}]$. Since fig. 6.5 shows that models with learning rates outside of the interval $[1 \times 10^{-3}, 1 \times 10^{-2}]$ perform significantly worse, the probability to sample a suitable learning rate is only 25%. A different reason for the bad performance of the GNNs in the GeneralizedAggressive AHPO could be that the generalized architectures are inferior to the ParticleNet-like architectures used in section 5.3. It could be argued that the EdgeConv blocks appended after the peak width was reached introduced a bottleneck in the GNN, i.e., the many node features learned by the EdgeConv block at the peak width had to be subsequently compressed into a representation with fewer nodes. This compression could decrease the amount of information the GNN could use in the subsequent dense layers to make a final prediction. Unfortunately, no statistically significant statement about these effects can be made from the data collected during this AHPO. Since one of the main goals of the AHPOs is improving the GNNs' performance, it was decided to terminate the AHPOs using the generalized architectures to focus on ParticleNet-like architectures which are already shown to perform well. This decision was made quickly after one week to ensure enough time remains to collect sufficient data for statistically significant statements.

6.4.2. ParticleNet-like architectures

Defining the hyperparameter space

The hyperparameter ranges and the parameters configuring the ASHA scheduler for the two AHPOs performed with ParticleNet-like architectures can be seen in listing A.3. The hyperparameter ranges are shared between both AHPOs, making comparisons much more straightforward. The main difference between the two AHPOs is that the first trains on the training set of the merged dataset with intermediate weighting, where $\alpha = 1$ and $\beta = 0.5$, while the second AHPO trains on the analogously weighted training set of the simulation group dataset. The second AHPO was performed because the computing time budget of 46 000 GPU-hours could not be fully consumed in time for the first AHPO, which allowed performing a parallel second AHPO to confirm the results of section 5.2, stating that the merged dataset is more suitable for tau identification than the simulation group dataset, with more statistics. Analogous to the naming convention established in section 6.4.1, the first AHPO will be referred to as “ParticleNetMerged” from now on. In contrast, the second AHPO will be called “ParticleNetSimGroup”.

Analogous to the hyperparameter ranges, the parameters configuring the ASHA scheduler are also shared between both AHPOs, except for the `grace_period`. The ParticleNetMerged AHPO has a `grace_period` of two, while the ParticleNetSimGroup uses a `grace_period` of one, which was selected to increase the statistics of the ParticleNetSimGroup AHPO because it ran for a shorter time than the ParticleNetMerged AHPO. Note that the `grace_period` is reduced compared to the GeneralizedRelaxed AHPO since it was concluded in the last section that an aggressive approach is more feasible.

Many hyperparameters defining the ParticleNet-like architectures are shared with the generalized architectures from section 6.4.1, whose definitions will not be repeated in this section. One new hyperparameter is the decay rate of the learning rate, which is nonzero for all sampled hyperparameter configurations. Analogous to the last section, a learning rate decay happens exponentially. The decay rate determines by which factor the learning rate decayed after the maximum number of 50 epochs is reached. Another difference between the ParticleNet-like architectures and the generalized architectures is that the ParticleNet-like architectures do not add decreasing EdgeConv blocks after the peak width is reached. Hence, only one hyperparameter is sufficient to describe the number of EdgeConv blocks in the network. On the other hand, the number of dense layers is now determined by two hyperparameters, where the first parameter determines the number of dense layers after the global pooling that has a constant number of neurons. More specifically, the number of neurons in these dense layers equals the number of neurons in the kernel layers of the last EdgeConv block. The second parameter determines the number of dense layers with an exponentially decreasing number of neurons stacked behind the constant dense layers. The exponential increases and decreases are no longer limited to a factor of two. Instead, slight deviations are allowed by the `exponent_basis` hyperparameter, which determines by which factor the number of neurons in subsequent layers increases or decreases. The use of a floor function rounds multiplications that result in non-natural numbers. Finally, the three parameters β_1, β_2, and ϵ that configure the behavior of the Adam optimizer introduced in section 3.2 are now considered as variable hyperparameters.

The hyperparameter ranges of the ParticleNet-like architectures were not chosen by manual sampling from different hyperparameter spaces and searching for a suitable distribution. Instead, it was implemented into the `trainable` function of the AHPOs

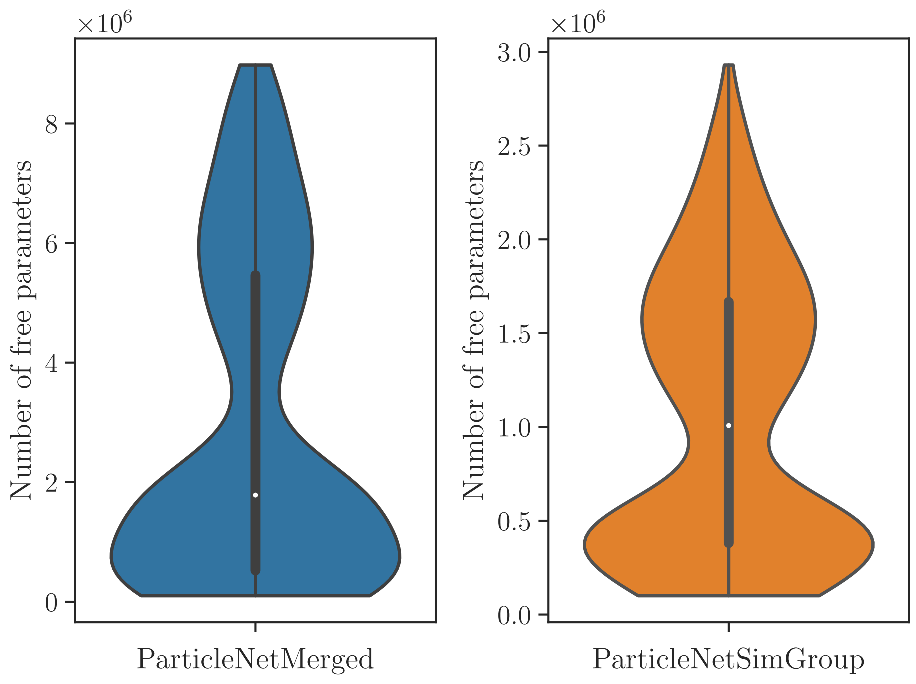


Figure 6.6.: The distribution of free parameters sampled from the hyperparameter space for the ParticleNet-like AHPOs.

that it should terminate models immediately if their number of free parameters lies outside a defined range. These models are stored as having a PR-AUC of zero on the validation set, which makes it possible to filter them easily before doing the analysis. This procedure allows the definition of many more combinations of hyperparameter ranges which were not possible before since they could result in too small or too large models. The range defining allowed values for the number of free parameters was defined as $[1 \times 10^5, 9 \times 10^6]$ for the ParticleNetMerged AHPO and $[1 \times 10^5, 3 \times 10^6]$ for the ParticleNetSimGroup AHPO, where the latter consists of considerably smaller values compared to the former and to the range used in section 6.4.1. This decision was made to reduce the average training time by removing GNNs with too many free parameters since the ParticleNetSimGroup AHPO had only few computing time. The distribution of the number of free parameters of these acceptable models is shown in fig. 6.6 for both AHPOs.

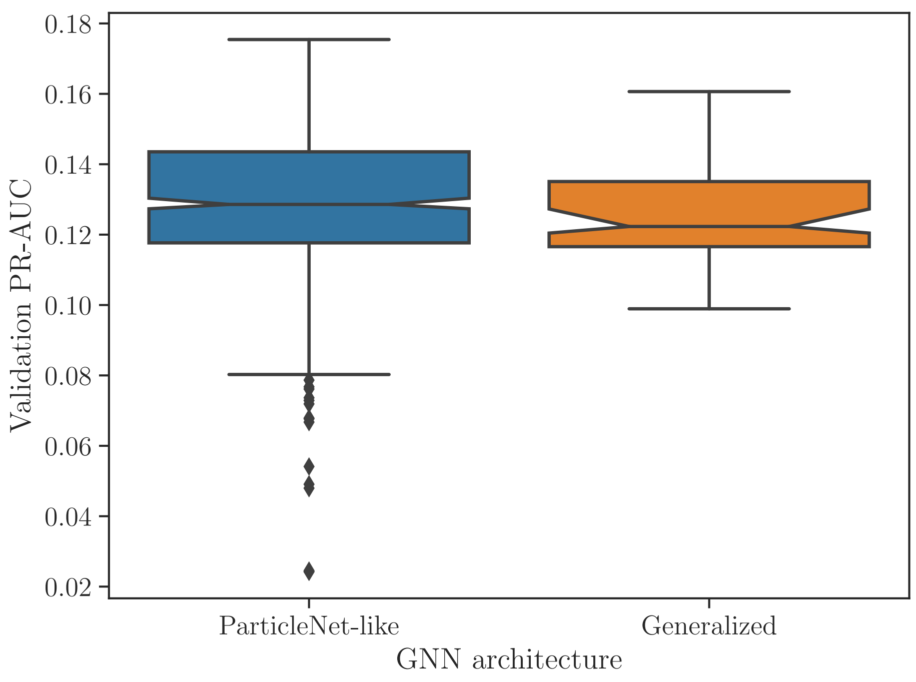


Figure 6.7.: Performance distributions of the architectures used in the GeneralizedAggressive and the ParticleNetMerged architectures. Only models with learning rates in the range $[1 \times 10^{-3}, 1 \times 10^{-2}]$ and a minimum training epoch of 2 are considered.

Analyzing the impact of different hyperparameters

The ParticleNetMerged AHPO resulted in a total of 1044 GNNs that lasted for 22 d, while the ParticleNetSimGroup AHPO trained 453 GNNs in 4 d due to its more aggressive value for the `grace_period`, its more constrained range of free parameters, and the reduced training time per epoch on its smaller training set.

A comparison between the performance of the generalized and the ParticleNet-like architectures is difficult because the hyperparameter ranges and the configuration of the ASHA scheduler differ significantly. Nevertheless, fig. 6.7 attempts to compare the two architectures by comparing the performances of models from the GeneralizedAggressive and the ParticleNetMerged AHPO. Since the ParticleNetMerged AHPO has the advantage that it only trained models with learning rates in the range $[1 \times 10^{-3}, 1 \times 10^{-2}]$, the comparison considers only models for the GeneralizedAggressive AHPO with learning rates in that range. Additionally, ParticleNetMerged has the advantage that its first elimination

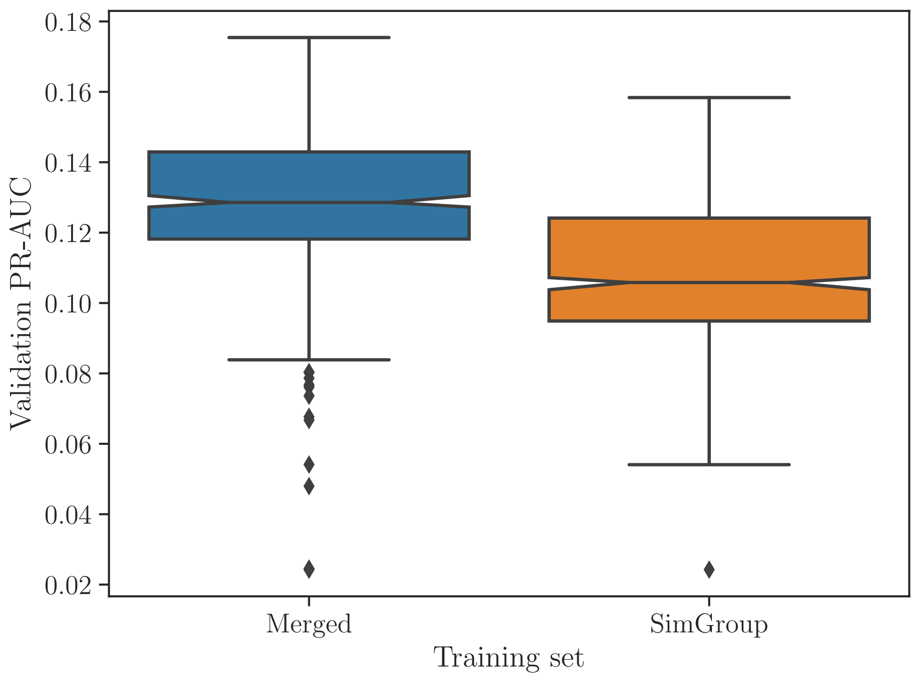


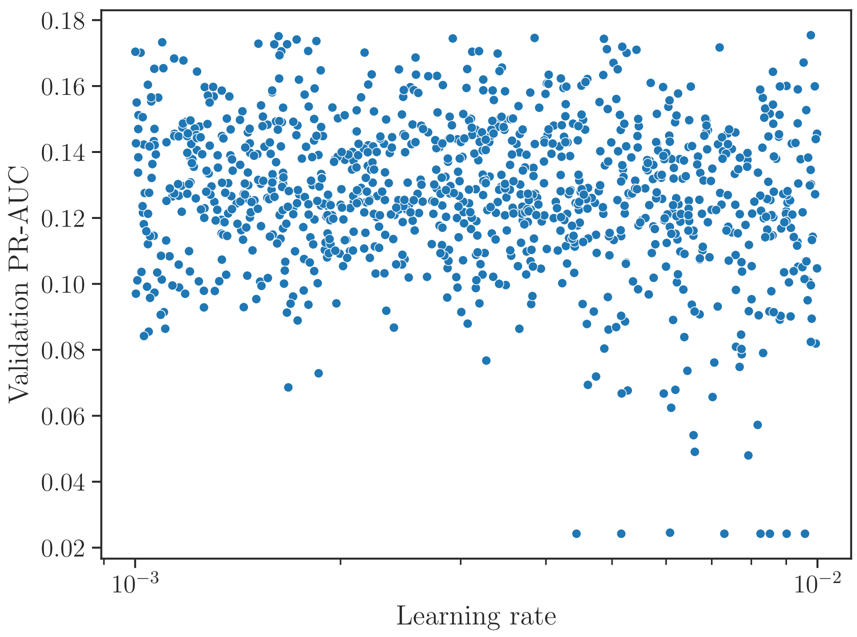
Figure 6.8.: Performance distributions of the ParticleNetMerged and the ParticleNet-SimGroup AHPOs, which differed mainly in their used training set. The models eliminated in the first termination round of the ParticleNetSimGroup AHPO were not considered. Only models with free parameters in the range $[10^5, 3 \cdot 10^6]$ are considered.

round happened after two epochs, unlike the GeneralizedAggressive AHPO, where models were already eliminated after one round. Hence, the comparison does not consider the models terminated after the first elimination round. Figure 6.7 shows that the median of the ParticleNet-like architectures is slightly better than for the GeneralizedAggressive AHPO. The best models trained in the ParticleNetMerged AHPO were better than the best models in the GeneralizedAggressive AHPO, showing comparable performance to the GNNs found during the manual hyperparameter optimization in section 5.3. Nevertheless, these well-performing models at the edges of the ParticleNet-like distribution in fig. 6.7 could be the result of more statistics combined with a better-constrained hyperparameter space in the case of the ParticleNetMerged AHPO. Hence, no definitive conclusion can be drawn about the superiority of one GNN architecture over the other.

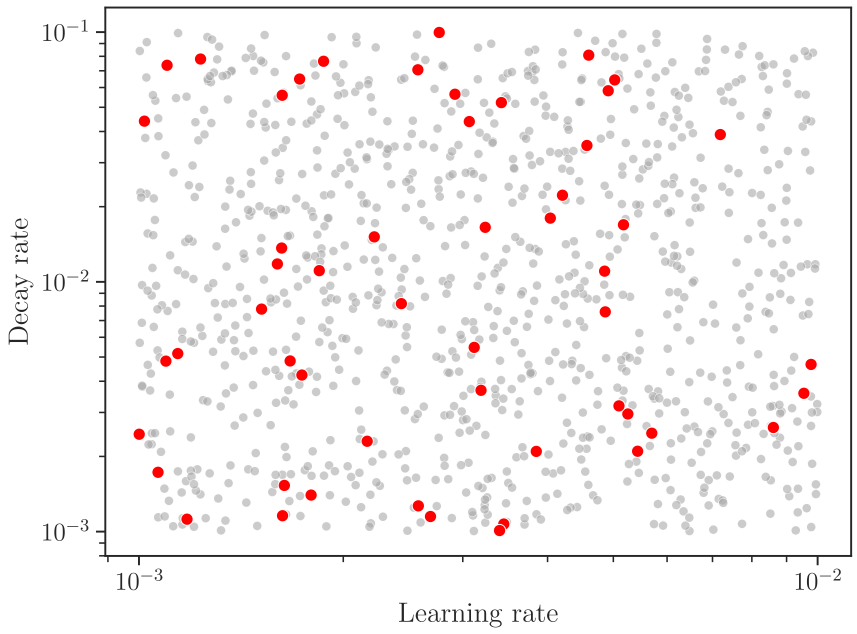
A comparison between the ParticleNetMerged and the ParticleNetSimGroup AHPO is much simpler since both architectures share the same hyperparameter space and, except for the `grace_period`, the same parameters configuring the ASHA scheduler. The ParticleNetMerged AHPO has an advantage over the ParticleNetSimGroup AHPO since it does not terminate bad-performing models in the first epoch. Hence, the models terminated in the first elimination round in the MergedSimGroup AHPO are not considered for the comparison, which gives an advantage to the MergedSimGroup AHPO since its worst-performing models are ignored. Furthermore, the ParticleNetMerged AHPO allowed models with more than $3 \cdot 10^6$ free parameters that are filtered out for a better comparison. The performance distributions of the resulting comparison can be seen in fig. 6.8, which confirms that the training data produced for the merged dataset in chapter 4 led to a significant improvement of the GNNs' performance. The plot also shows that the best-performing models of the ParticleNetSimGroup AHPO reach performances comparable to the best model found in section 5.2.4, which indicates that the model used in that section was already well-optimized for training on the simulation group dataset. Since the data collected from the ParticleNetMerged AHPO has more statistics and better performance than the ParticleNetSimGroup AHPO, the rest of this section will only use data collected from that AHPO.

The relationship between the sampled learning rates and the model performances is shown in fig. 6.9a. The plot indicates that learning rates close to 1×10^{-2} might have a slightly higher probability of producing models with bad performance. A significant relationship between the best-performing models and the learning rate can not be identified in the plot. Figure 6.9b shows the combinations of the sampled learning and decay rates. The 5% of models with the highest PR-AUC on the validation set are shown in red. Again, no significant relationship between the learning and decay rates can be concluded for the best-performing models.

Figure 6.10a shows the performance distributions for models with a different number of neurons in the kernel layers of their first EdgeConv block. The plot shows that models with only 32 neurons in their first kernel layers show significantly worse performance than models with more neurons. A significant conclusion for an increasing number of neurons in the first kernel layers can not be drawn, although there seems to be a slight upwards trend for the median. A possible reason for this trend could be that a small number of neurons in the first kernel layer creates an information bottleneck. The network is forced to update all nodes with the same kernel network. If the number of neurons in the first layer of this shared kernel network is too small, the network may not be able to extract all information from the raw node features, leading to an impaired performance.



(a) The achieved performances of the models versus their sampled learning rate.



(b) The combinations of learning rates and decay rates sampled from the hyperparameter space. The best 5% of trained models are labeled with red points.

Figure 6.9.

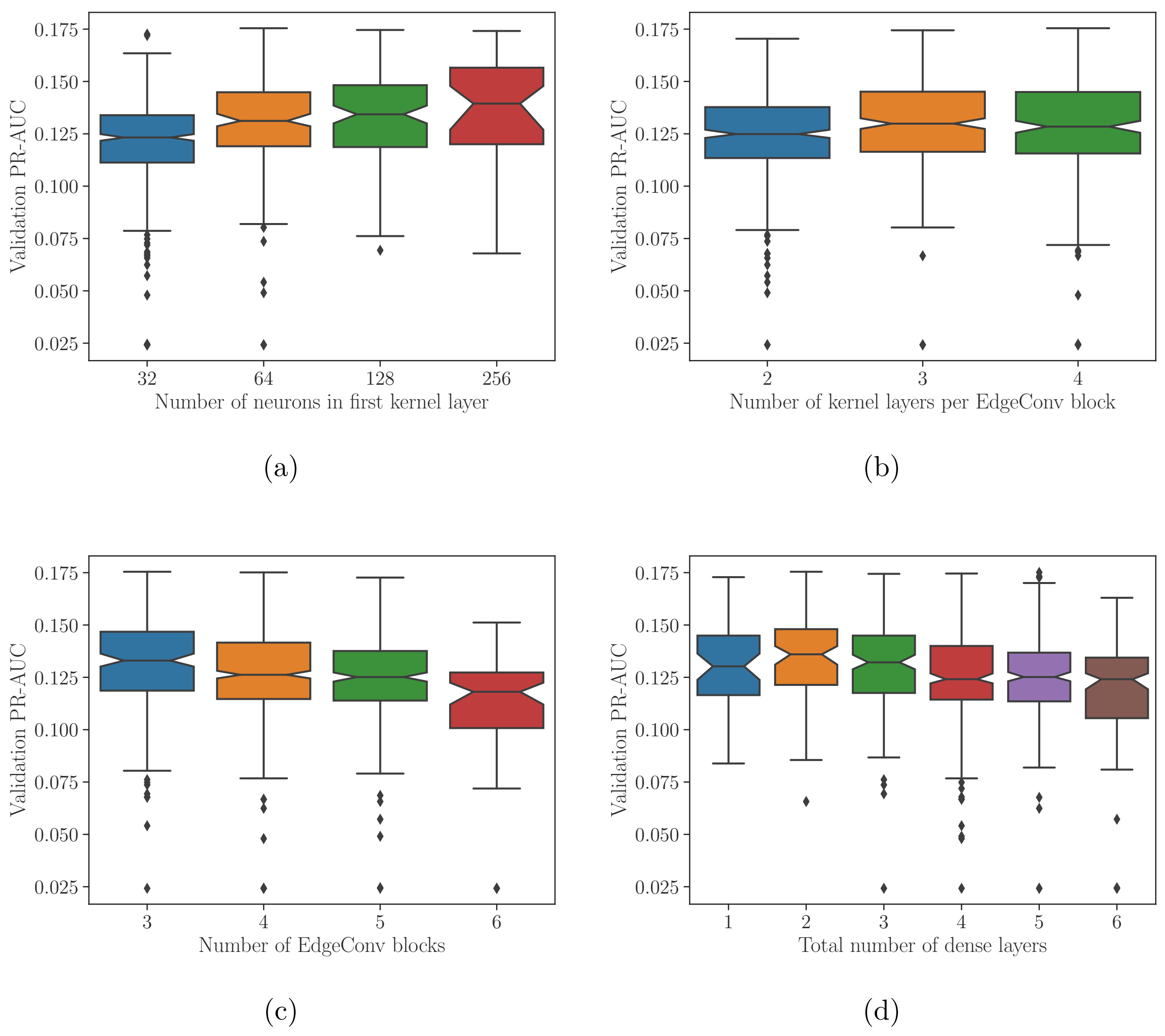
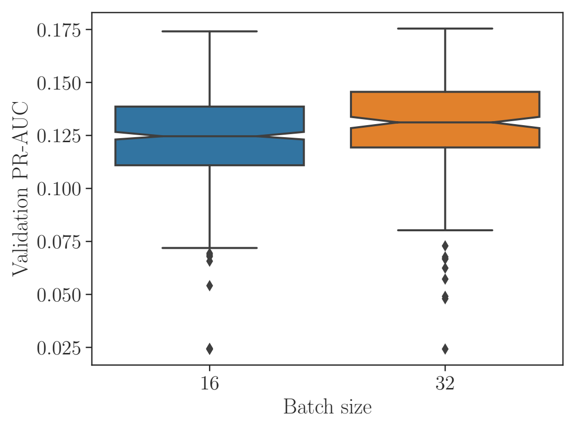
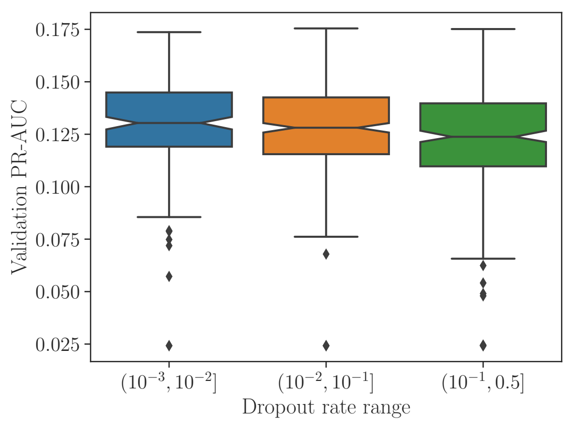


Figure 6.10.: Performance distributions resulting from variations in the network architecture.

Furthermore, fig. 6.10b shows a significant decrease in performance if the number of kernel layers per EdgeConv block is chosen too small. There are no hints of increasing performance for values larger than three. Figure 6.10c shows that models with three EdgeConv blocks perform significantly better than models with more EdgeConv blocks. A possible reason for this behavior could be that a model with too many EdgeConv blocks has a large numbers of free parameters causing it overfit too quickly on the data. The performance of larger models is expected to increase with the availability of more training data. These three results support the choices made in the design of the ParticleNet architecture shown in fig. 3.7. Figure 6.10d shows the performance of



(a)



(b)

Figure 6.11.: Performance distributions for different batch sizes and dropout rate ranges.

models with different values for the total number of dense layers, i.e., the added number of constant and decreasing dense layers. The plot shows hints of a slightly increased performance for models with a total number of two dense layers, contrary to the decision made in ParticleNet, which uses only one dense layer. These four hyperparameters were the main parameters responsible for the number of free parameters in the models. In total, fig. A.4a shows that no significant dependency of the model performance on the number of free parameters could be found.

Figure 6.11a shows a significant increase in the model performance for a batch size of 32 compared to a smaller batch size of 16. The reason for this behavior is probably that larger batch sizes result in gradient updates that are more precise since the loss function was approximated better. Figure 6.11b shows the performance distributions of models with different dropout rates. There seems to be a slight downward trend in the performance for increasing dropout rates. Although the ParticleNet architecture in fig. 3.7 uses dropout for its last dense layer, removing the dropout for the GNNs trained on KM3NeT data may be advisable. This result supports the decisions in [18, 19], where the dropout was removed from the ParticleNet-like architectures.

One of the most surprising results of this thesis is that the ϵ parameter of the Adam optimizer, which is included for numerical stability, is one of the most significant hyper-parameters in the model. The performance distributions resulting from a variation of ϵ are shown in fig. 6.12, which shows a significant decrease in the model performance for values deviating from 0.1. As discussed in section 3.2, the value for the ϵ parameter

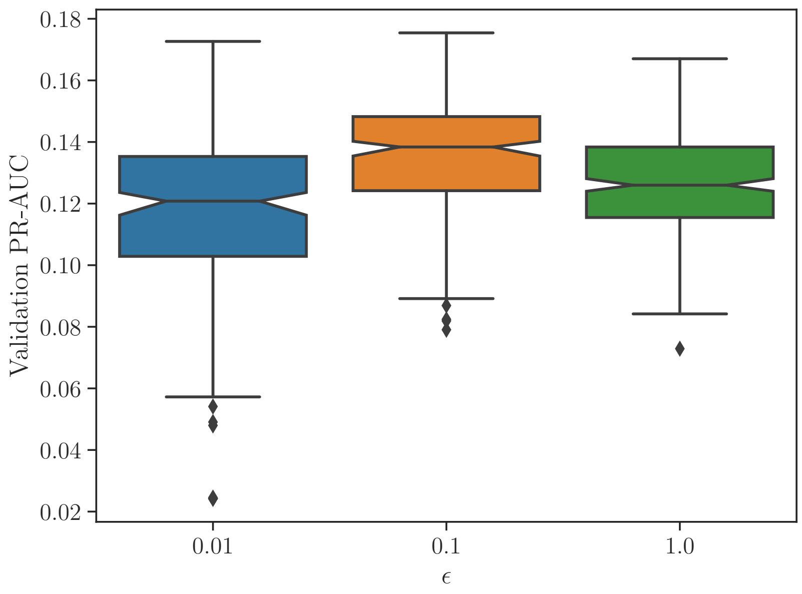


Figure 6.12.: Performance distributions resulting from a variation of the ϵ parameter in the Adam optimizer.

recommended in [22] is 10^{-8}. However, the documentation in TensorFlow[23] states that the default choice is not optimal for all models. Hence, setting the default value of ϵ in OrcaNet to 0.1 was an excellent decision.

The other hyperparameters varied in the ParticleNetMerged AHPO did not result in significant changes in the GNNs' performances or did not show any unexpected results. Hence, they are not discussed in detail in this section but are appended in appendix A.5.

6.5. Evaluation of the best-performing network

The last section of this chapter will evaluate the model that reached the highest PR-AUC on the validation set of the Honda weighted simulation group dataset. The best model found during the AHPOs is a model with a PR-AUC of 0.175, which is slightly higher than the PR-AUC of the best-performing model found in the manual hyperparameter optimization, which has a score of 0.174. The best-performing model was found during the ParticleNetMerged AHPO, and its hyperparameter configuration is appended in listing A.4.

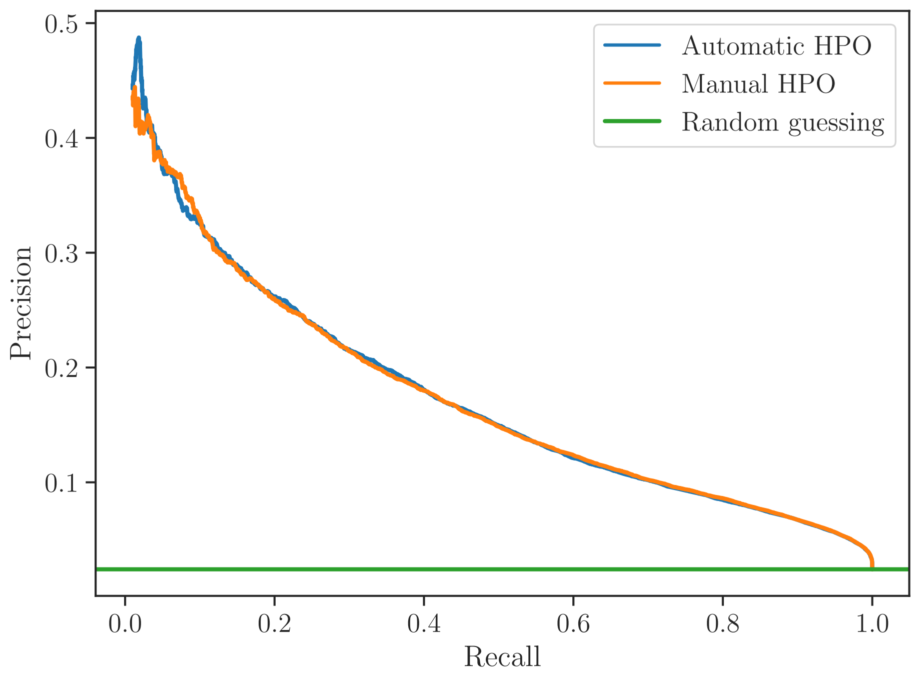


Figure 6.13.: Precision-recall curves of the best model from the automatic and the manual hyperparameter optimization on the validation set.

Figure 6.13 shows the precision-recall curves of the best-performing model from the automatic and the manual hyperparameter optimization. Both models perform very similar for recalls larger than 10%. The precision-recall curves of both models deviate for smaller recalls, which could be the effect of random statistical fluctuations. A decreasing recall is equivalent to a decreasing number of positively labeled tau events, which causes the small recall values to be more susceptible to random statistical fluctuations. These statistical fluctuations influence the area under the precision-recall curve, which was used as a performance metric throughout this thesis. Since the best-performing models only deviate in the third decimal place in the values of their performance metric, it has to be assumed that a statistically significant statement about the actual best model found in this thesis can not be made. However, a statistical analysis of the effects of random statistical fluctuations on the performance metric is beyond the scope of this thesis. Hence, for the rest of this thesis, the best-performing model is defined to be the model that reached the highest value in the chosen performance metric.

As briefly discussed in section 4.1, training thousands of models with subsequent evaluations on the same validation set could lead to models that perform well on the validation

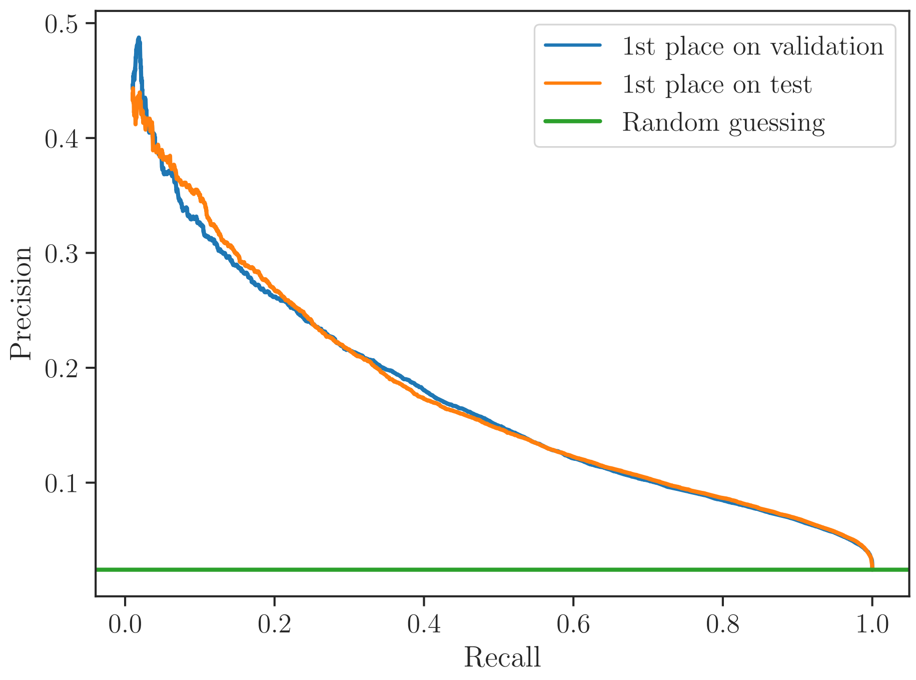


Figure 6.14.: Precision-recall curves of the best-performing model evaluated on the validation and test set.

set just by chance. Hence, a test set consisting of 15% of the total events in the simulation group dataset is used for a final evaluation of the best-performing model. The test set consists of events that were neither used during training nor validation so far, which ensures that the model is not optimized on that set of events. The result of the evaluation on the test set is shown in fig. 6.14. Since the model was optimized for the validation set, it is expected that the model has a similar or better performance on the validation set than on the test set. This behavior can be seen for most recalls above 20%. However, the model performs better on the test set than the validation set for recalls smaller than 20%. This behavior could, again, be caused by random statistical fluctuations. In addition, it could also be suspected that the model performs better on the test set because the events in the test set are easier to classify, which would introduce a systematic uncertainty to the problem. This suspicion arose after reevaluating the process by which the whole dataset was divided into a training, validation, and test set. A good procedure to split a large dataset into three smaller homogenous sets is to concatenate all neutrino event files from all runs and interaction types into a list that gets thoroughly shuffled before being split up into the three subsets. However, this procedure is typically not done in KM3NeT since

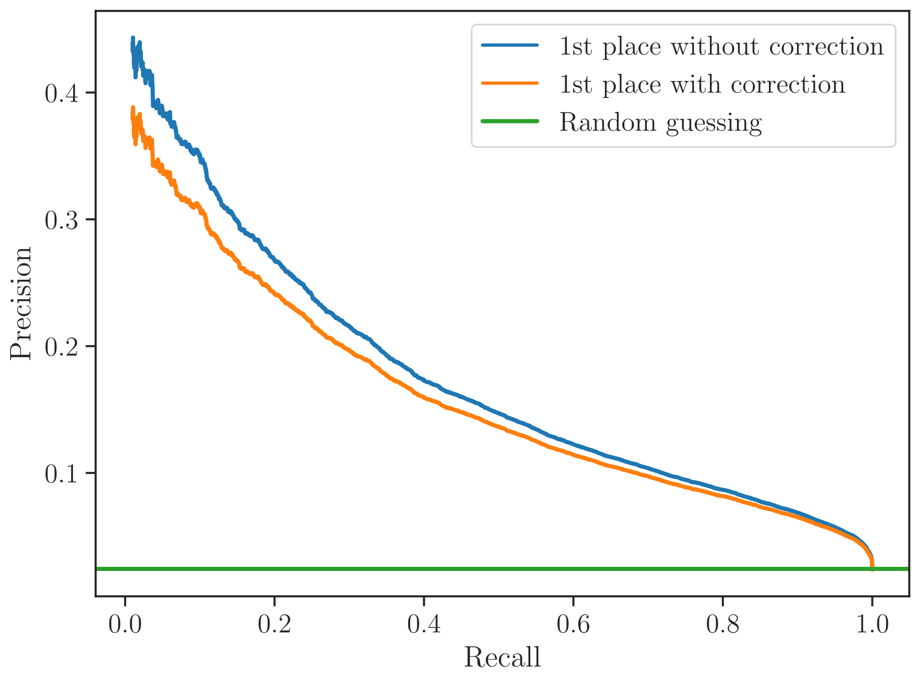


Figure 6.15.: Precision-recall curves of the best-performing model on the Honda weighted test set with and without the correction of the bug.

shuffling a complete list of all neutrino events would cause memory problems. Instead, the three subsets are often created by randomly assigning entire files (r, γ) to the three subsets, where r is the run ID of the file, and γ is the interaction type simulated in the file. This procedure was also followed while creating the simulation group dataset's training, validation, and test set. Since the procedure of randomly assigning entire files (r, γ) to the three subsets is much more susceptible to causing inhomogeneities in the three subsets than shuffling the complete list of neutrino events, the test set may contain some runs that are easier to classify for the GNNs because, e.g., the runs were simulated during time periods with reduced background activities in the seawater. However, this suspected systematic uncertainty can only be confirmed or rejected after a more detailed analysis, which is beyond the scope of this thesis.

As discussed in section 5.1.1, the Honda weighting used in this thesis contained a bug that caused an underrepresentation of neutral-current events visualized in fig. 5.1. The precision-recall curve of the best-performing model on the test set with corrected Honda weighting is shown in fig. 6.15. The correction causes a deterioration of the model's performance, which is expected since the GNN is optimized to perform well on a differently

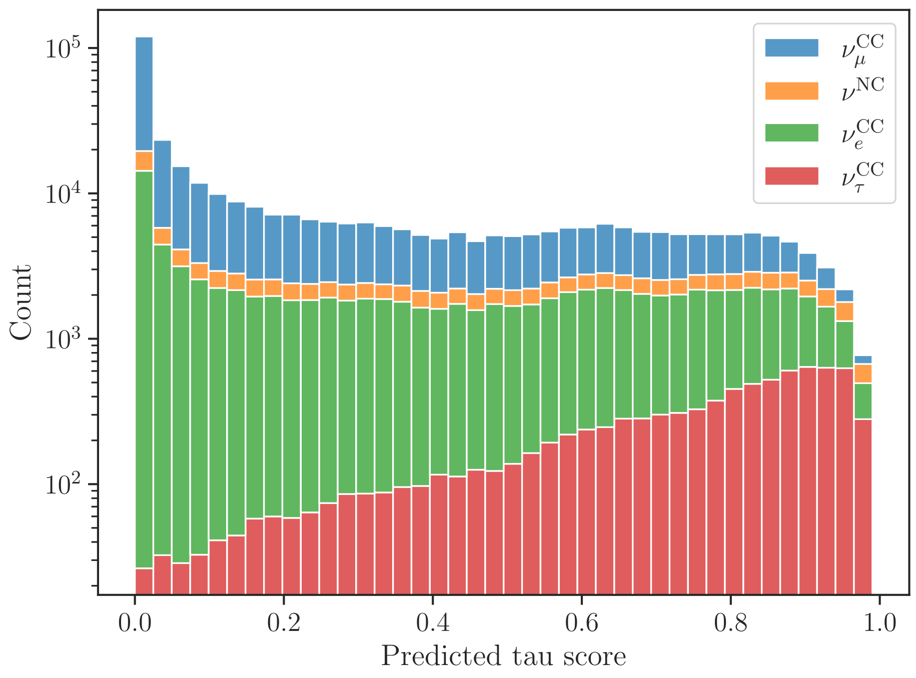


Figure 6.16.: The tau scores predicted by the best-performing model on the corrected Honda weighted test set. Note the logarithmic scale on the y-axis. The colored bins corresponding to events with particular interaction types are stacked on each other. Each interaction type includes particle as well as antiparticle interactions.

weighted test set. Nevertheless, the plot shows that the GNN is still able to reach a precision of 19.7% at a recall of 30% on the correctly weighted test set. As shown in fig. 5.1, the original fraction of tau events in the dataset is 2.4%, demonstrating that the GNN is able to increase the fraction of tau events by a factor of more than eight to 19.7% while still identifying 30% of the tau events correctly.

The final plot of this thesis shows the distribution of predicted tau scores of the best-performing GNN in fig. 6.16. The predictions were made on the test set of the simulation group dataset, which was weighted with the corrected Honda weights. The increasing height of the red bins in the plot shows that the GNN successfully learned to predict higher tau scores for tau events. On the other hand, bins corresponding to smaller tau scores are mostly filled with nontau events, showing that the GNN learned a set of features to filter these events out.

7. Conclusion

7.1. Summary

The thesis starts with a discussion of neutrino properties with a focus on the phenomenon of neutrino oscillations and the PMNS matrix. This complex-valued matrix is assumed to be unitary in the current established neutrino oscillation model, which is a property that can be tested by further constraining the PMNS matrix element with neutrino oscillation experiments. The analysis of the transition probabilities of atmospheric neutrinos created in the muon flavor and oscillated into the tau flavor contributes to achieving that goal. These analyses need neutrino event datasets enriched with tau events, making it necessary to develop methods able to produce tau-enriched datasets. This thesis introduces a new approach to tackle the problem of tau-enrichment by training Graph Neural Networks to directly identify tau neutrino events in a neutrino event dataset.

The Graph Neural Networks are trained on Monte Carlo (MC) simulations of events simulated in the KM3NeT/ORCA detector, which is a neutrino detector currently under construction in the Meditarranean sea. While the full detector will consist of 115 so-called detection units, this thesis uses simulations from an early stage of the detector consisting of six detection units.

Chapter 3 gives a short introduction to Deep Learning, leading to the discussion of Graph Neural Networks (GNNs), which are neural networks taking graphs as input. The chapter explains how neutrino events in KM3NeT/ORCA are represented as graphs, making it possible to train GNNs on simulated neutrino data to directly identify tau events.

Chapter 4 introduces the available neutrino event dataset, in which tau events make up a fraction of only 16.5%. The limited statistics of tau neutrino events led to the decision to produce additional MC simulations for this thesis. The MC data was produced with a tool developed as part of this thesis, which allows the production of triggered MC events following an arbitrary energy spectrum. Although the tool is still in a development stage, training the GNNs on a combination of the existing and the additionally produced data led to a significant performance improvement.

Chapter 5 consists of manual optimization steps aiming to find the optimal training set and optimal values for the GNNs' hyperparameters. A comparison of different training sets composed of events from different MC productions is performed. Additionally, different sample weights to alter the distributions of physical quantities in the datasets are tried out. The latter is done by introducing a parametrized weighting method determining how the sample weights are calculated. Different values for the parameters of the weighting method are evaluated, resulting in a training set that increases the performance of the GNNs on the validation set significantly. This dataset is subsequently used to train a limited number of GNNs in a manual hyperparameter optimization, also resulting in a performance increase.

The final chapter of this thesis introduces the concept of automatic hyperparameter optimization (AHPO), which is a modern approach to the problem of hyperparameter optimization. The AHPOs are performed using the "Asynchronous Successive Halving Algorithm" (AHSA)[50] which is pre-implemented in Ray Tune[46]. The chapter explains in detail how an AHPO can be configured using Ray Tune. Four different AHPOs are performed using a computing time budget of 46 000 GPU-hours, in which different GNN architectures and hyperparameter spaces are evaluated. The AHPOs resulted in GNNs reaching higher performance metrics than the GNN from the manual hyperparameter optimization, but the differences in their performances are too small to arrive at a statistically significant performance ranking. In total, the AHPOs trained 2064 GNNs, which were used to identify the hyperparameters with the highest impact on the GNNs' performance. The analysis confirms most of the choices for hyperparameter values made in previous studies using GNNs on KM3NeT/ORCA's data, showing that the currently used GNNs are well optimized. Furthermore, it was demonstrated that the GNN with the highest performance metric can increase the fraction of tau events by a factor of eight to about 20% while still correctly identifying 30% of all tau neutrino events in the simulated dataset.

7.2. Outlook

Chapter 4 demonstrated that producing additional Monte Carlo (MC) simulations is a powerful method to improve the GNNs' performance. The MC data for this thesis was produced with flat energy spectra, which was not the most suitable choice since the models performed best on a dataset with intermediate weighting, where $\alpha = 1$ and $\beta = 0.5$. Hence, it might be recommended to produce triggered MC events following the energy spectra in fig. 5.2, which could be achieved with the tool implemented in this thesis. Additionally, the tool could be extended to allow the production of datasets with arbitrary zenith distributions, such that training sets could be produced following the energy spectra and zenith distributions of intermediate weighting with $\alpha = 1$ and $\beta = 0.5$. Finally, the tool could be combined with the run-by-run MC simulation pipeline[37] that was implemented in Snakemake, eliminating its current constraint that it can only produce events for one single run.

The AHPOs performed in this thesis successfully found GNNs reaching improved performance metrics over the manual hyperparameter optimization, but the choice of the PR-AUC as the performance metric was not optimal in hindsight due to the statistical fluctuations in the precision-recall curve for small recall values. A better version of the PR-AUC might be a metric like the "PR-AUC-X", which could be defined as the area under the precision-recall curve for recall values larger than X%. Suitable values for X can be chosen by analyzing the impact of statistical fluctuations on the area under the precision-recall curve. The AHPO confirmed most of the decisions made in the ParticleNet architecture, which shows that this architecture is already well-optimized. AHPOs can be considered powerful tools for finding suitable hyperparameters for new network architectures on which no prior experience exists. However, performing an AHPO on an already well-established network architecture might not lead to significant performance improvements. The usage of the ASHA algorithm can be recommended if the scheduler is configured to terminate bad trials sufficiently aggressively. It is recommended to start with a large hyperparameter space and to check the results of an AHPO regularly to identify the most significant hyperparameters early. Early identification of suitable values for a significant hyperparameter allows one to constrain the range of that hyperparameter to these suitable values, drastically reducing the number of bad-performing models. This procedure could also be automated by choosing a more sophisticated search algorithm in Ray Tune than the random search used in this thesis.

This thesis aims to study whether GNNs can be trained to directly identify tau neutrino events in KM3NeT/ORCA's data such that they can be used to produce a dataset enriched with tau neutrinos. The currently established approach of enriching a dataset with tau events consists of separating track and shower events. This indirect approach works because most of the nontau events result from ν_μ^{CC} interactions that produce track events, which are filtered away into a track class by a track-shower-classifier. Additionally, as explained in section 2.3, more than 80% of tau events result in shower events, implying that most tau neutrinos are assigned to the shower class by the classifier. Hence, the shower class resulting from that approach is enriched with tau events. In contrast, this thesis' direct approach of identifying tau events searches explicitly for features in the training data that make a discrimination between tau and nontau events possible. One of the next steps should be to test whether tau-enriched datasets produced with the direct approach lead to better sensitivities than those reached with the established indirect approach.

7.3. Acknowledgements

I want to use this section to thank all of my colleagues at ECAP and KM3NeT for the amazing time that I had during my master's thesis! I'm very thankful for the many insightful discussions about neutrino physics and machine learning during our meetings and lunch breaks! Special thanks go to

- **PD Dr. Thomas Eberl** for giving me the opportunity to write this thesis and for always providing great support whenever I had questions!
- **Prof. Uli Katz** for his support throughout the whole course of my physics studies! Both Thomas and you provide a great working environment for us students!
- **Nicole Geißelbrecht and Rodrigo Gracia-Ruiz** for our many insightful discussions and for always having answers to my questions!
- **Tamás Gál** for always helping me with software-related questions!
- **Johannes Veh** from the NHR@FAU for supporting me with the application for computing resources!
- **Philip Kaludercic, Michael Chadolias, and Bastian Setter** for reviewing this thesis!

The author gratefully acknowledges the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) under the NHR project b173dc. NHR funding is provided by federal and Bavarian state authorities. NHR@FAU hardware is partially funded by the German Research Foundation (DFG) – 440719683.

Bibliography

1. Pauli, W. Dear radioactive ladies and gentlemen. *Phys. Today* **31N9,** 27 (1978).
2. Fermi, E. Versuch einer Theorie der β-Strahlen. I. *Zeitschrift für Physik* **88,** 161–177. ISSN: 0044-3328. `https://doi.org/10.1007/BF01351864` (Mar. 1934).
3. Reines, F., Cowan, C. L., Harrison, F. B., McGuire, A. D. & Kruse, H. W. Detection of the Free Antineutrino. *Phys. Rev.* **117,** 159–173. `https://link.aps.org/doi/10.1103/PhysRev.117.159` (1 Jan. 1960).
4. MissMJ & Cush. *Standard Model of Elementary Particles* `https://en.wikipedia.org/w/index.php?lang=en&title=File%3AStandard_Model_of_Elementary_Particles.svg` (2023).
5. Fukuda, Y. *et al.* Evidence for Oscillation of Atmospheric Neutrinos. *Phys. Rev. Lett.* **81,** 1562–1567. `https://link.aps.org/doi/10.1103/PhysRevLett.81.1562` (8 Aug. 1998).
6. Ahmad, Q. R. *et al.* Direct Evidence for Neutrino Flavor Transformation from Neutral-Current Interactions in the Sudbury Neutrino Observatory. *Phys. Rev. Lett.* **89,** 011301. `https://link.aps.org/doi/10.1103/PhysRevLett.89.011301` (1 June 2002).
7. *The Nobel Prize in Physics 2015* `https://www.nobelprize.org/prizes/physics/2015/summary/` (2023).
8. Maki, Z., Nakagawa, M. & Sakata, S. Remarks on the Unified Model of Elementary Particles. *Progress of Theoretical Physics* **28,** 870–880. ISSN: 0033-068X. eprint: `https://academic.oup.com/ptp/article-pdf/28/5/870/5258750/28-5-870.pdf`. `https://doi.org/10.1143/PTP.28.870` (Nov. 1962).
9. Pontecorvo, B. Neutrino Experiments and the Problem of Conservation of Leptonic Charge. *Zh. Eksp. Teor. Fiz.* **53,** 1717–1725 (1967).

10. Esteban, I., Gonzalez-Garcia, M. C., Maltoni, M., Schwetz, T. & Zhou, A. The fate of hints: updated global analysis of three-flavor neutrino oscillations. *Journal of High Energy Physics* **2020,** 178. ISSN: 1029-8479. `https://doi.org/10.1007/JHEP09(2020)178` (Sept. 2020).

11. *NuFIT 5.2* `www.nu-fit.org` (2023).

12. Katz, U. & Spiering, C. High-energy neutrino astrophysics: Status and perspectives. *Progress in Particle and Nuclear Physics* **67,** 651–704. ISSN: 0146-6410. `https://www.sciencedirect.com/science/article/pii/S0146641011001189` (2012).

13. Kajita, T. Atmospheric neutrinos and discovery of neutrino oscillations. en. *Proc. Jpn. Acad. Ser. B Phys. Biol. Sci.* **86,** 303–321 (2010).

14. Hallmann, S. Sensitivity to atmospheric tau-neutrino appearance and all-flavour search for neutrinos from the Fermi Bubbles with the deep-sea telescopes KM3NeT/ORCA and ANTARES. en. `https://ecap.nat.fau.de/wp-content/uploads/2021/02/Dissertation_HallmannSteffen_Opus.pdf` (2021).

15. Evidence for High-Energy Extraterrestrial Neutrinos at the IceCube Detector. *Science* **342.** `https://doi.org/10.1126/science.1242856` (Nov. 2013).

16. Adrián-Martínez, S. *et al.* Letter of intent for KM3NeT 2.0. *Journal of Physics G: Nuclear and Particle Physics* **43,** 084001. `https://doi.org/10.1088/0954-3899/43/8/084001` (June 2016).

17. Tanabashi, M. *et al.* Review of Particle Physics. *Phys. Rev. D* **98,** 030001. `https://link.aps.org/doi/10.1103/PhysRevD.98.030001` (3 Aug. 2018).

18. Guderian, D. Development of detector calibration and graph neural network-based selection and reconstruction algorithms for the measurement of oscillation parameters with KM3NeT/ORCA. en. `https://www.uni-muenster.de/imperia/md/content/physik_kp/agkappes/abschlussarbeiten/doktorarbeiten/doktorarbeit_daniel_guderian.pdf` (2022).

19. Reck, S. Cosmic ray composition measurement using Graph Neural Networks for KM3NeT/ORCA. en (2023).

20. Neutelings, I. `https://tikz.net/neural_networks/` (2023).

21. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* `http://www.deeplearningbook.org` (MIT Press, 2016).

22. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2017. arXiv: 1412.6980 [cs.LG].

23. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam#notes_2 (2023).

24. Ioffe, S. & Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* 2015. arXiv: 1502.03167 [cs.LG].

25. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **15,** 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html (2014).

26. Wang, Y. *et al.* Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* **38.** ISSN: 0730-0301. https://doi.org/10.1145/3326362 (Oct. 2019).

27. Qu, H. & Gouskos, L. Jet tagging via particle clouds. *Physical Review D* **101.** https://doi.org/10.1103%2Fphysrevd.101.056019 (Mar. 2020).

28. Reck, S., Moser, M., Guderian, D. & Gal, T. *OrcaNet* en. 2021. https://zenodo.org/record/6631956.

29. Reck, S. *MEdgeConv* https://github.com/StefReck/MEdgeConv (2023).

30. Aiello, S. *et al.* gSeaGen: The KM3NeT GENIE-based code for neutrino telescopes. *Computer Physics Communications* **256,** 107477. https://doi.org/10.1016%2Fj.cpc.2020.107477 (Nov. 2020).

31. *KM3Sim GitLab repository* https://git.km3net.de/simulation/km3sim (2023).

32. *Jpp GitLab repository* https://git.km3net.de/common/jpp (2023).

33. *OrcaSong GitLab repository* https://git.km3net.de/ml/OrcaSong (2023).

34. Mölder, F. *et al.* Sustainable data analysis with Snakemake. *F1000Research* **10** (2021).

35. Yoo, A. B., Jette, M. A. & Grondona, M. *SLURM: Simple Linux Utility for Resource Management* in *Job Scheduling Strategies for Parallel Processing* (eds Feitelson, D., Rudolph, L. & Schwiegelshohn, U.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 2003), 44–60. ISBN: 978-3-540-39727-4.

36. Hennig, L. https://git.km3net.de/workflow-management/snakemake-workflows/-/tree/neutrino-simulations-flat-spectra-smk-on-node?ref_type=heads (2023).

37. Pestel, V. `https://git.km3net.de/vpestel/snakemake-workflows/-/tree/main/data_processing` (2023).

38. Honda, M., Athar, M. S., Kajita, T., Kasahara, K. & Midorikawa, S. Atmospheric neutrino flux calculation using the NRLMSISE-00 atmospheric model. *Physical Review D* **92.** `https://doi.org/10.1103%2Fphysrevd.92.023004` (July 2015).

39. *HKKM2014 flux tables* `https://www.icrr.u-tokyo.ac.jp/~mhonda/nflx2014/index.html` (2023).

40. Lotze, M. & Gal, T. *km3flux* `https://git.km3net.de/km3py/km3flux` (2023).

41. Coelho, J. *OscProb* `https://github.com/joaoabcoelho/OscProb` (2023).

42. Gal, T. *km3services* `https://git.km3net.de/km3py/km3services` (2023).

43. O'Malley, T. *et al. KerasTuner* `https://github.com/keras-team/keras-tuner`. 2019.

44. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. *Optuna: A Next-generation Hyperparameter Optimization Framework* in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019).

45. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830 (2011).

46. Liaw, R. *et al.* Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118* (2018).

47. *Ray* `https://www.ray.io/` (2023).

48. *Ray Tune FAQ* `https://docs.ray.io/en/latest/tune/faq.html` (2023).

49. *AHPO for tau identification GitLab repository* `https://git.km3net.de/lhennig/automatic_hpo_gnns_tauid_lukas` (2023).

50. Li, L. *et al. Massively Parallel Hyperparameter Tuning* 2018. `https://openreview.net/forum?id=S1Y7OOlRZ`.

51. Jamieson, K. G. & Talwalkar, A. Non-stochastic Best Arm Identification and Hyperparameter Optimization. *CoRR* **abs/1502.07943.** arXiv: `1502.07943`. `http://arxiv.org/abs/1502.07943` (2015).

52. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization* 2018. arXiv: `1603.06560 [cs.LG]`.

A. Appendix

A.1. Spectra and distributions characterizing datasets

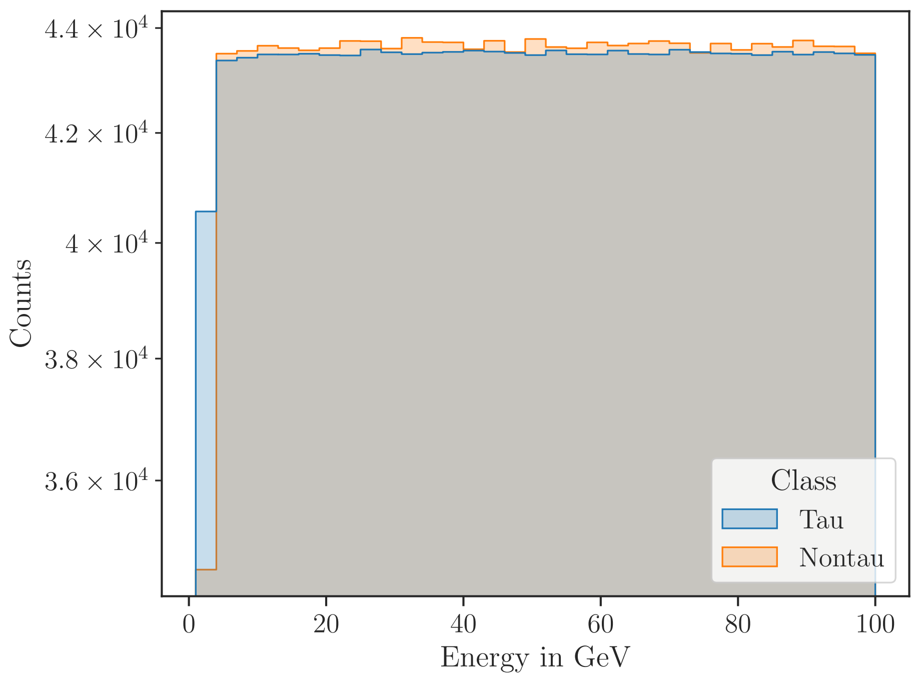


Figure A.1.: The energy spectrum of the flat dataset. The simulations were produced in the energy range from 1 GeV to 100 GeV with energy bins of width 3 GeV.

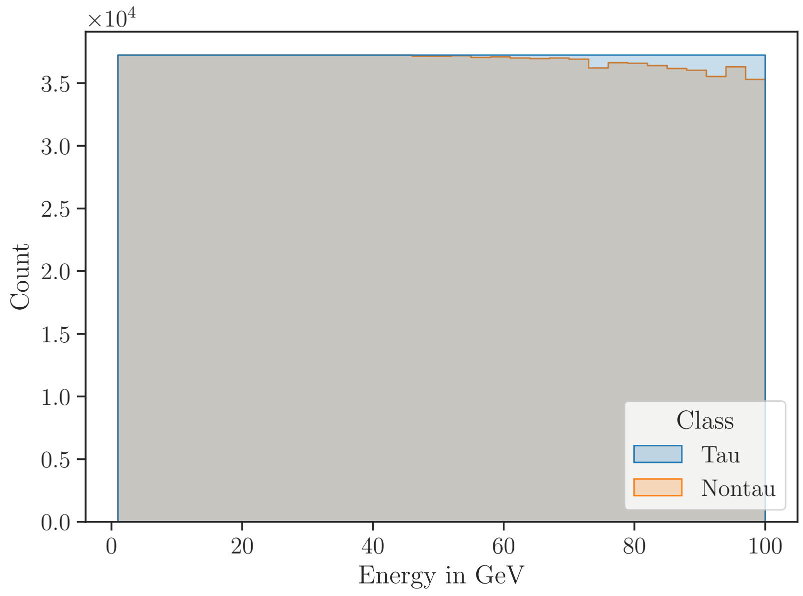


Figure A.2.: The energy spectra of the tau and nontau class for a flat weighted simulation group dataset.

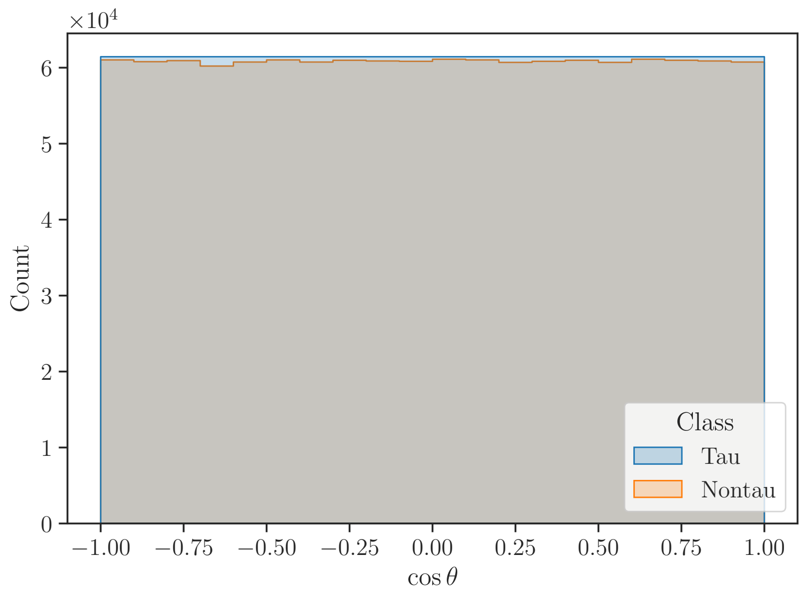


Figure A.3.: The $\cos\theta$ distributions of the tau and nontau class for a flat weighted simulation group dataset.

A.2. OrcaNet configuration files

Toml files configuring the GNN in section 5.2. The other GNNs in chapter 5 were configured analogous.

Config

```
[config]
batchsize = 16
learning_rate = [ 0.001, 0.0,]
train_logger_display = 100
train_logger_flush = -1
verbose_train = 2
shuffle_train = true
cleanup_models = true
dataset_modifier = "as_array"
use_custom_sample_weights = true

[config.sample_modifier]
name = "GraphEdgeConv"
knn = 30

[config.label_modifier]
name = "ClassificationLabels"
column = "particle_type"
model_output = "tau_nontau_scores"

[config.label_modifier.classes]
class1 = [ 16, -16,]
```

Model

```
[model]
type = "DisjointEdgeConvBlock"
next_neighbors = 30
shortcut = true
[[model.blocks]]
units = [ 64, 64, 64,]
batchnorm_for_nodes = true
```

```

[[model.blocks]]
units = [ 128, 128, 128,]

[[model.blocks]]
units = [ 256, 256, 256,]
pooling = true

[[model.blocks]]
type = "OutputCateg"
categories = 1
output_name = "tau_nontau_scores"
unit_list = [ 256,]

[compile]
optimizer = "adam"

[compile.losses.tau_nontau_scores]
function = "binary_crossentropy"

[compile.losses.tau_nontau_scores.weighted_metrics_classes.PR_AUC]
```

A.3. Detailed results of manual hyperparameter optimization

Batchsize	#Nearest neighbors	#EdgeConvs	#Dense	Best epoch	PR-AUC
16	30	3	2	29	0.174
16	30	5	1	21	0.174
32	30	4	1	24	0.174
16	30	5	2	21	0.173
16	30	4	1	22	0.173
32	30	5	3	18	0.172
32	30	4	2	21	0.172
16	30	4	2	27	0.172
32	20	5	2	15	0.171
16	20	4	2	21	0.171
16	30	3	1	29	0.171
16	20	5	3	19	0.170
32	20	5	3	18	0.170
16	20	5	2	16	0.170
32	30	5	1	20	0.169
16	20	4	1	18	0.169
32	30	5	2	16	0.169
32	20	4	1	17	0.169
16	30	5	3	22	0.168
32	20	4	2	18	0.167
32	20	5	1	19	0.166
16	20	5	1	14	0.166

Table A.1.: The results of the manual hyperparameter optimization in descending order sorted by the PR-AUC. Best epoch refers to the epoch in which the GNN performed best on the weighted validation set of the merged dataset, while the PR-AUC is evaluated on the Honda weighted validation set of the simulation group dataset.

A.4. Parameters defining the automatic hyperparameter optimizations

```python
from ray import tune
import numpy as np


class ParamSpace:
    def __init__(self) -> None:
        self.learning_rate = tune.qloguniform(1e-5, 1e-1, 5e-6)
        self.batchsize = tune.choice([16, 32])
        self.shortcut = tune.choice([True, False])
        self.batchnorm = tune.choice([True, False])
        self.useDropout = tune.choice([True, False])
        self.dropout_rate = tune.sample_from(get_dropout_rate)
        self.kNN = tune.choice([30, 35, 40, 45, 50])


class ParamSpacePerBlock(ParamSpace):
    def __init__(self) -> None:
        super().__init__()
        self.number_kernel_network_nodes_begin = tune.choice([24, 26, 28, 30])
        self.number_kernel_layers = tune.choice([2, 3, 4])
        self.number_edgeconvs_until_peak = tune.choice([5, 6])
        self.number_edgeconvs_after_peak = tune.sample_from(
            lambda spec: np.random.randint(0, spec.config.number_edgeconvs_until_peak)
        )
        self.number_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                spec.config.number_edgeconvs_until_peak
                - spec.config.number_edgeconvs_after_peak,
            )
        )


class ParamSpacePerKernelLayer(ParamSpace):
    def __init__(self) -> None:
        super().__init__()
        self.number_kernel_network_nodes_begin = tune.choice([8, 10,
    12, 16, 36, 56])
        self.number_kernel_layers = tune.choice([2])
        self.number_edgeconvs_until_peak = tune.sample_from(
            get_number_edgeconvs_until_peak_for_per_kernel_layer
        )
        self.number_edgeconvs_after_peak = tune.sample_from(
            lambda spec: np.random.randint(0, 3)
        )
        self.number_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                3 - spec.config.number_edgeconvs_after_peak,
            )
        )


def get_dropout_rate(spec):
    if spec.config.useDropout:
        return tune.quniform(5e-2, 7e-1, 5e-3)
    else:
        return None


def get_number_edgeconvs_until_peak_for_per_kernel_layer(spec):
    if spec.config.number_kernel_network_nodes_begin < 15:
        return tune.choice([4])
    else:
        return tune.choice([3])


class ASHAConfig_GeneralizedAggressive:
    def __init__(self):
        self.time_attr = ("training_iteration",)
        self.max_t = (40,)
        self.grace_period = (1,)
        self.reduction_factor = (2,)
        self.brackets = (1,)
```

Listing A.1: Parameters defining the GeneralizedAggressive AHPO

```
class ParamSpace:
    def __init__(self) -> None:
```

```python
        self.learning_rate = tune.qloguniform(1e-5, 1e-1, 5e-6)
        self.use_lr_schedule = tune.choice([True, False])
        self.batchsize = tune.choice([16, 32])
        self.shortcut = tune.choice([True, False])
        self.batchnorm = tune.choice([True, False])
        self.useDropout = tune.choice([True, False])
        self.dropout_rate = tune.sample_from(get_dropout_rate)
        self.kNN = tune.choice([25, 30, 35, 40, 45])


class ParamSpacePerBlock(ParamSpace):
    def __init__(self) -> None:
        super().__init__()
        self.number_kernel_network_nodes_begin = tune.choice([24, 26,
    28, 30])
        self.number_kernel_layers = tune.choice([2, 3, 4])
        self.number_edgeconvs_until_peak = tune.choice([5, 6])
        self.number_edgeconvs_after_peak = tune.sample_from(
            lambda spec: np.random.randint(0, spec.config.
    number_edgeconvs_until_peak)
        )
        self.number_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                spec.config.number_edgeconvs_until_peak
                - spec.config.number_edgeconvs_after_peak,
            )
        )


class ParamSpacePerKernelLayer(ParamSpace):
    def __init__(self) -> None:
        super().__init__()
        self.number_kernel_network_nodes_begin = tune.choice([16, 24,
    30])
        self.number_kernel_layers = tune.choice([2, 3])
        self.number_edgeconvs_until_peak = tune.sample_from(
            get_number_edgeconvs_until_peak_for_per_kernel_layer
        )
        self.number_edgeconvs_after_peak = tune.sample_from(
            lambda spec: np.random.randint(0, 4)
        )
        self.number_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                4 - spec.config.number_edgeconvs_after_peak,
            )
        )


def get_dropout_rate(spec):
    if spec.config.useDropout:
        return tune.quniform(5e-2, 7e-1, 5e-3)
    else:
        return None


def get_number_edgeconvs_until_peak_for_per_kernel_layer(spec):
    if spec.config.number_kernel_layers == 2:
        return tune.choice([5])
    else:
        return tune.choice([4])


class ASHAConfig_GeneralizedRelaxed:
    def __init__(self):
        self.time_attr = ("training_iteration",)
        self.max_t = (50,)
        self.grace_period = (4,)
        self.reduction_factor = (2,)
        self.brackets = (3,)
```

Listing A.2: Parameters defining the GeneralizedRelaxed AHPO

```
class ParamSpace:
    def __init__(self) -> None:
        self.learning_rate = tune.loguniform(1e-3, 1e-2)
        self.decay_rate = tune.loguniform(1e-3, 1e-1)
        self.batchsize = tune.choice([16, 32])
        self.shortcut = tune.choice([True, False])
        self.batchnorm = tune.choice([True, False])
        self.dropout_rate = tune.loguniform(1e-3, 5e-1)
        self.kNN = tune.choice([25, 30, 35, 40])
        self.number_kernel_network_nodes_begin = tune.choice([32, 64,
    128, 256])
```

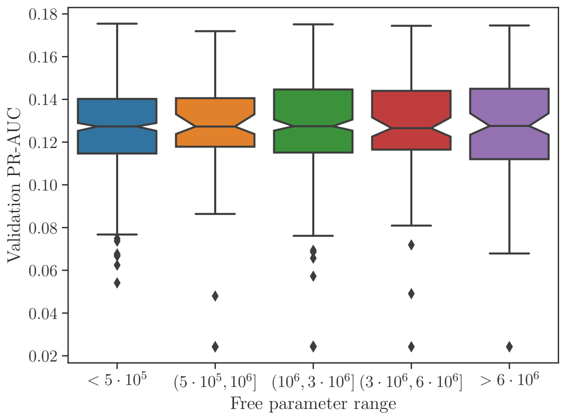
```python
        self.number_kernel_layers = tune.choice([2, 3, 4])
        self.number_edgeconvs = tune.choice([3, 4, 5, 6])
        self.number_constant_dense_layers = tune.choice([1, 2, 3])
        self.number_decreasing_dense_layers = tune.sample_from(
            lambda spec: np.random.randint(
                0,
                spec.config.number_edgeconvs,
            )
        )
        self.exponent_basis = tune.choice([1.9, 1.95, 2, 2.05])
        self.beta_1 = tune.choice([0.88, 0.89, 0.9, 0.91, 0.92])
        self.beta_2 = tune.choice([0.9988, 0.9989, 0.999, 0.9991,
    0.9992])
        self.epsilon = tune.choice([0.01, 0.1, 1.0])


class ASHAConfig_ParticleNetLikeMerged:
    def __init__(self):
        self.time_attr = ("training_iteration",)
        self.max_t = (50,)
        self.grace_period = (2,)
        self.reduction_factor = (2,)
        self.brackets = (3,)


class ASHAConfig_ParticleNetLikeSimGroup:
    def __init__(self):
        self.time_attr = ("training_iteration",)
        self.max_t = (50,)
        self.grace_period = (1,)
        self.reduction_factor = (2,)
        self.brackets = (3,)
```

Listing A.3: Parameters defining the ParticleNet-like AHPOs

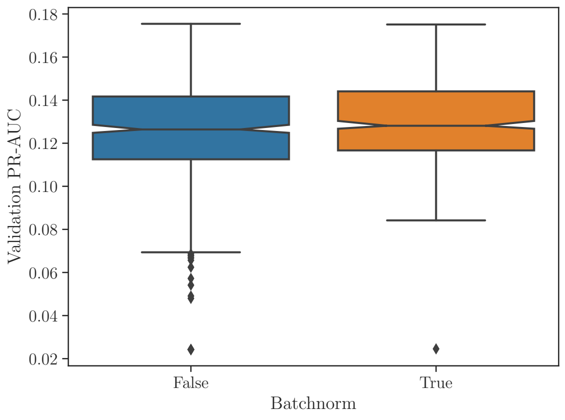
A.5. Further results from the automatic hyperparameter optimizations

```
{
    "batchnorm": False,
    "batchsize": 32,
    "beta_1": 0.89,
    "beta_2": 0.9992,
    "decay_rate": 0.004671360618855324,
    "dropout_rate": 0.010027171360068445,
    "epsilon": 0.1,
    "exponent_basis": 2,
    "kNN": 30,
    "learning_rate": 0.009780042164109768,
    "number_constant_dense_layers": 1,
    "number_decreasing_dense_layers": 1,
    "number_edgeconvs": 3,
    "number_kernel_network_nodes_begin": 64,
    "number_kernel_units": 4,
    "shortcut": True,
}
```

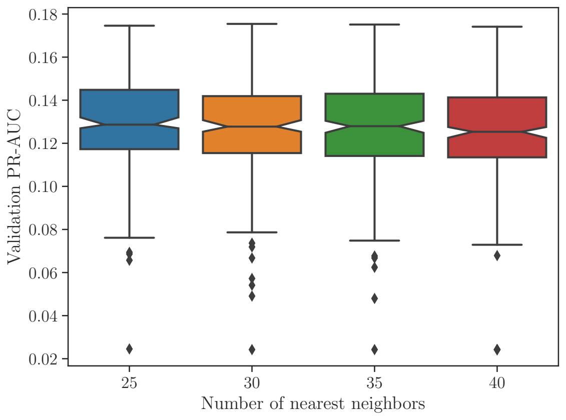
Listing A.4: Hyperparameter configuration of the best perfoming model with a PR-AUC of 0.175



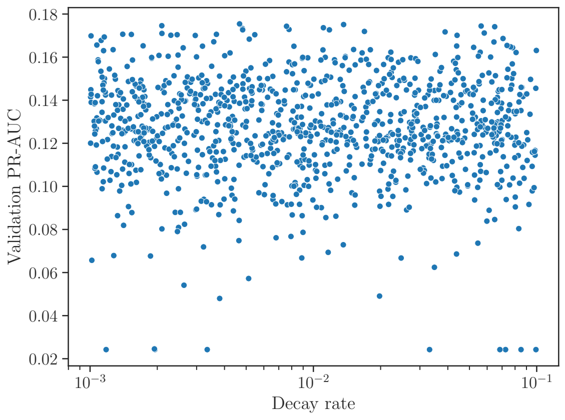
(a)



(b)

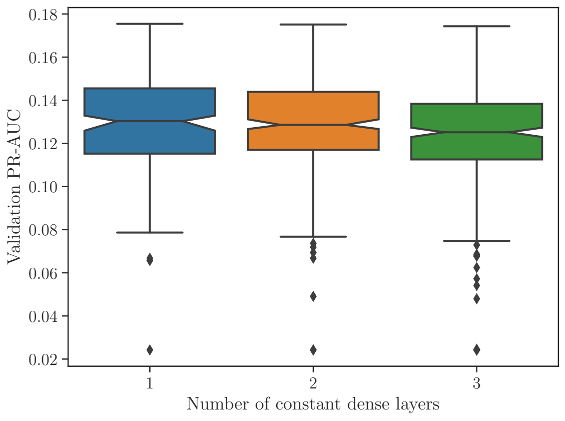


(c)

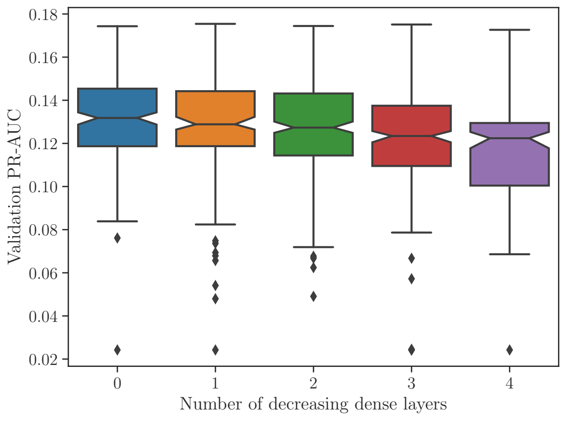


(d)

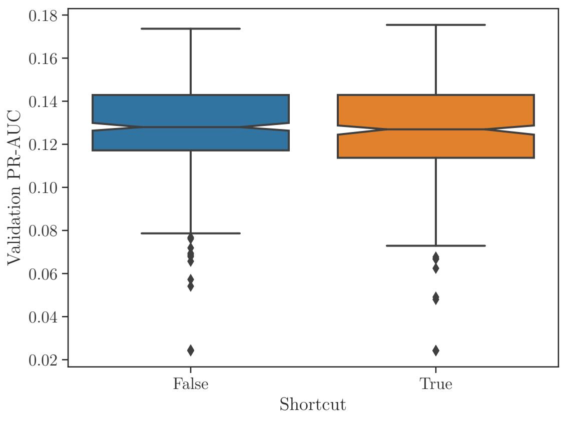
Figure A.4.



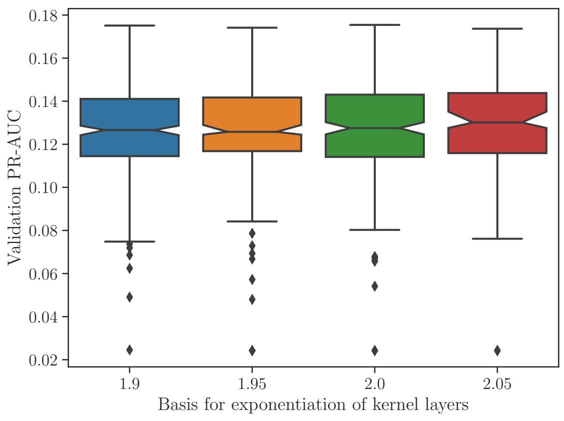
(a)



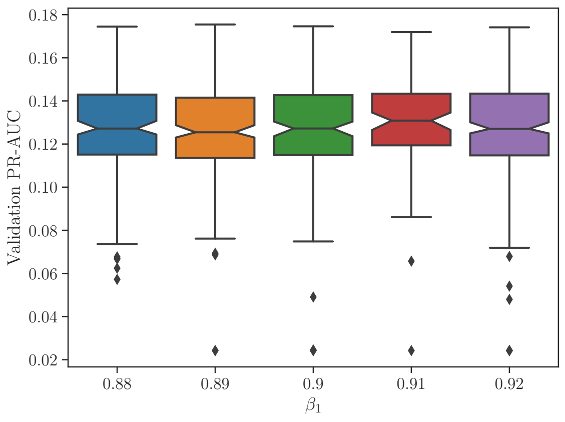
(b)



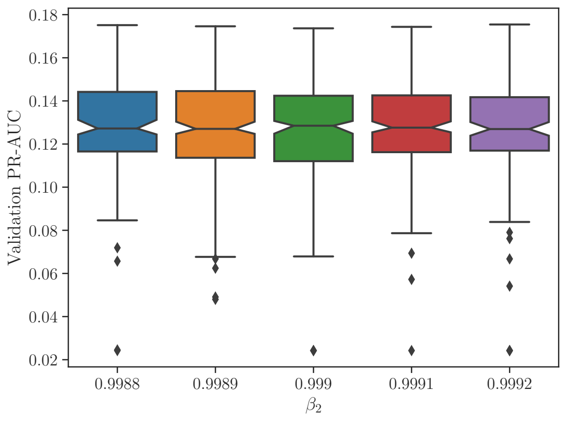
(c)



(d)



(e)



(f)

Figure A.5.

Statutory Declaration

I, Lukas Hennig, hereby declare that I wrote this thesis entirely by myself and did not use any sources or tools other than the ones referenced.

Erlangen, June 1, 2023