Analysis Specific Filters for Smart Background Simulations at Belle II

Luca Schinnerl

Master′s Thesis
at the Faculty of Physics
Ludwig–Maximilians–Universität München
Chair of Elementary Particle Physics

Supervisors:
Prof. Dr. Thomas Kuhr
Dr. Nikolai Hartmann

Munich, August 8th, 2022

Analysespezifische Filter für intelligente Untergundsimulation bei Belle II

Luca Schinnerl

Masterarbeit
an der Fakulität für Physik
Ludwig–Maximilians–Universität München
Lehrstuhl für Experimentelle Flavorphysik

Betreuer:
Prof. Dr. Thomas Kuhr
Dr. Nikolai Hartmann

München, den 8. August, 2022

Abstract

The Belle II experiment is expected to accumulate a data sample of 50ab^{-1} in its lifetime. For rare processes, strong background suppression is needed to precisely measure these types of events. Because of this, an extremely large number of simulated background events is necessary for an effective analysis. However, a significant portion of the simulated data is discarded trivially in the first stage of analysis, demanding a better method of simulation to keep up with the amount of data. For this purpose a neural network is implemented to select the relevant data after the Monte Carlo event generation and then only run the costly detector simulation and reconstruction for selected events. Existing methods have shown good success with graph neural networks. However, the total speedup of simulations is limited when considering generic selections. Here a maximum speedup of the simulation with the smart background filter over the brute force method of 2.1 was reached. In this work I iteratively introduce analysis specific filters to the training of the neural networks, which can greatly increase efficiencies. For an analysis selection in the search for the rare process $B \to K^{*}vv$ this methodology has been successful in significantly improving simulation speed.

Contents

Chapter 1

The Belle II Experiment

In high energy physics (HEP), physicists work with particle accelerators to analyze specific processes that are occurring when particles interact with each other. Our current best theory of the universe, the Standard Model, is unable to explain a large variety of different experimental observations including but not limited to:

- the strong CP problem [17];
- neutrino oscillations [28];
- matter–antimatter asymmetry [19];
- the nature of dark matter and dark energy [22].

All of these observation hint to physics beyond the Standard Model, so called new physics (NP). In the search for NP, e^+e^- colliders have shown decades long success with the Belle [3] and BaBar [10] experiments. In particular, the Belle experiment was able to confirm the Kobayashi-Maskawa mechanism [31]. This discovery led to the Nobel Prize in physics in 2008. The upgraded version of the Belle experiment, the Belle II experiment, is expected to collect a lot more data, approximately 50ab^{-1}. The Standard Model does not provide sufficient sources for CP violation to explain the matter –antimatter asymmetry [19]. Therefore, the hope of the Belle II experiment is among others to provide a better understanding of new sources of CP violation [47].

There are several reasons why experimentalists might favor using B-factories like Belle II for their analysis, instead of for example colliding protons as is done in the Large Hadron Collider (LHC [14]). The most important reason being that B-factories, in contrast to hadronic colliders, branching fractions can directly be measured. This is because the number of initially produced B particles is known to a very small uncertainty. Hadronic colliders are only able to measure ratios of branching fractions. The physical process that makes this possible is a special resonance that can be achieved when colliding a e^+e^- pair at a very specific energy (resonance energy). This is a so called Υ meson [9]. In the case of Belle II, the 4th resonance is utilized $\Upsilon(4S)$ as shown in figure 1.1. Electron–positron collisions at the resonance energy decay in the vast majority of times into B meson pairs, where the initial energy is well known. This enables the possibility for a full reconstruction of all the particles, including the processes with missing final states such as neutrinos. With this, high precision measurements are possible. Even analysis of rare events, events with a branching fraction of 10^{-6} or less, are possible. This provides sensitivity to NP in rare processes that are forbidden at tree–level in the Standard Model [26].

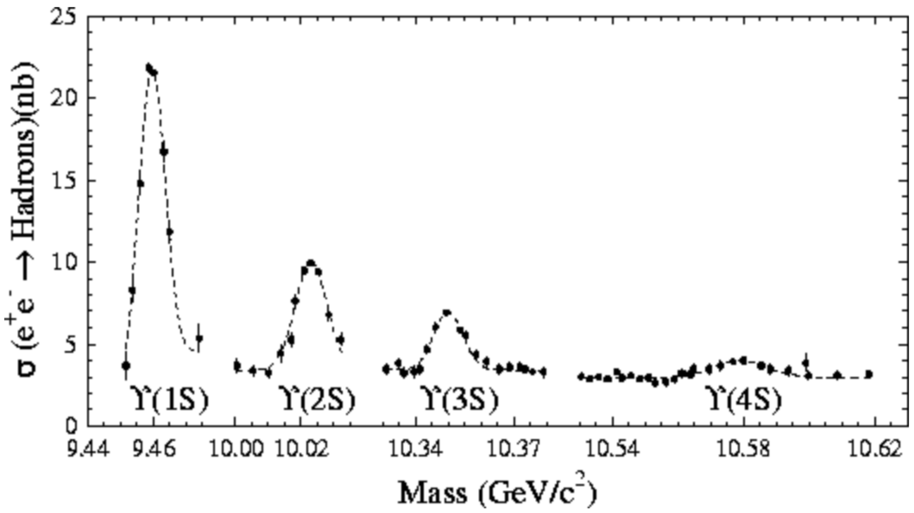


Figure 1.1: Υ resonances and different energies. At Belle II the center of mass energy is approximately 10.58GeV to produce the 4th resonance.

1.1 SuperKEKB

The Belle II experiment is based at the Japanese High-Energy Accelerator Research Organization (KEK) in Japan. It will use SuperKEKB as its assymetric-energy e^-e^+ double ring collider. It was constructed by upgrading the existing KEKB B-Factory [48], with the goal of massively increasing the luminosity [39]. The SuperKEKB collider is comprised of

- a 7−GeV electron ring, the blue high energy ring as shown in 1.2;
- a 4−GeV positron ring, the red low energy ring as shown in 1.2;
- an electron-positron injector linear accelerator (linac);
- a 1.1−GeV positron damping ring (DR) to achieve resonance of the Υ meson [30];
- and finally the Belle II detector itself [4].

The improvements to the target luminosity of $6.5 \times 10^{35} cm^{-2}s^{-1}$, 30 times larger than the luminosity of the KEKB collider, can be partly attributed to an increased beam current

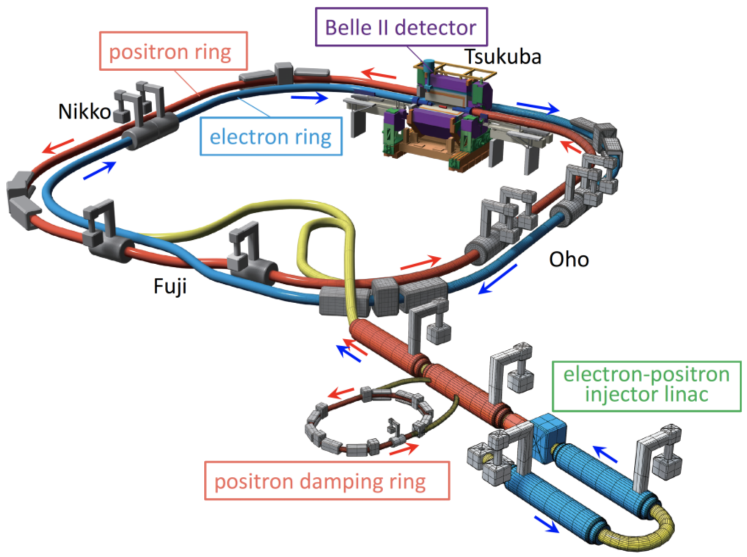


Figure 1.2: Schematic overview of the SuperKEKB collider. The electron and positron beams collide at the interaction point in the Tsukuba straight section where the Belle II detector is located.

and mostly to the so called 'nano–beam' scheme [7]. In this scheme, particle beams are squeezed in the vertical direction and the crossing angle is increased, reducing the area of crossing and increasing the luminosity [39]. The history of luminosity is shown in figure 1.3, showing a clear increase with each run.

1.2 The Belle II Detector

At the heart of the Belle II experiment lies the Belle II detector [4] [36]. The detector itself is a refurbished version of the Belle detector, where the main design challenge is the amount of data taking it needs to be able to handle (40 times higher luminosity [5]). In other words, it has to be able to operate at 40 times higher physics event rates, as well as with the corresponding background rates, which are higher by a factor of 10 to 20. A labeled schematic drawing of this is shown in figure 1.5. To achieve optimal performance, the detector is comprised out of a range of different subdetectors:

- Vertex detector: Measure vertices of the two B meson decays. It includes two subdetectors, a pixel detector (PIX) and a silicon vertex detector (SVD).
- Central drift chamber (CDC): measures the exact trajectories of charged particles to reconstruct their momenta, to perform particle identification in the low-momentum region using energy loss in the CDC volume. Furthermore, it also generates trigger information for charged particles.

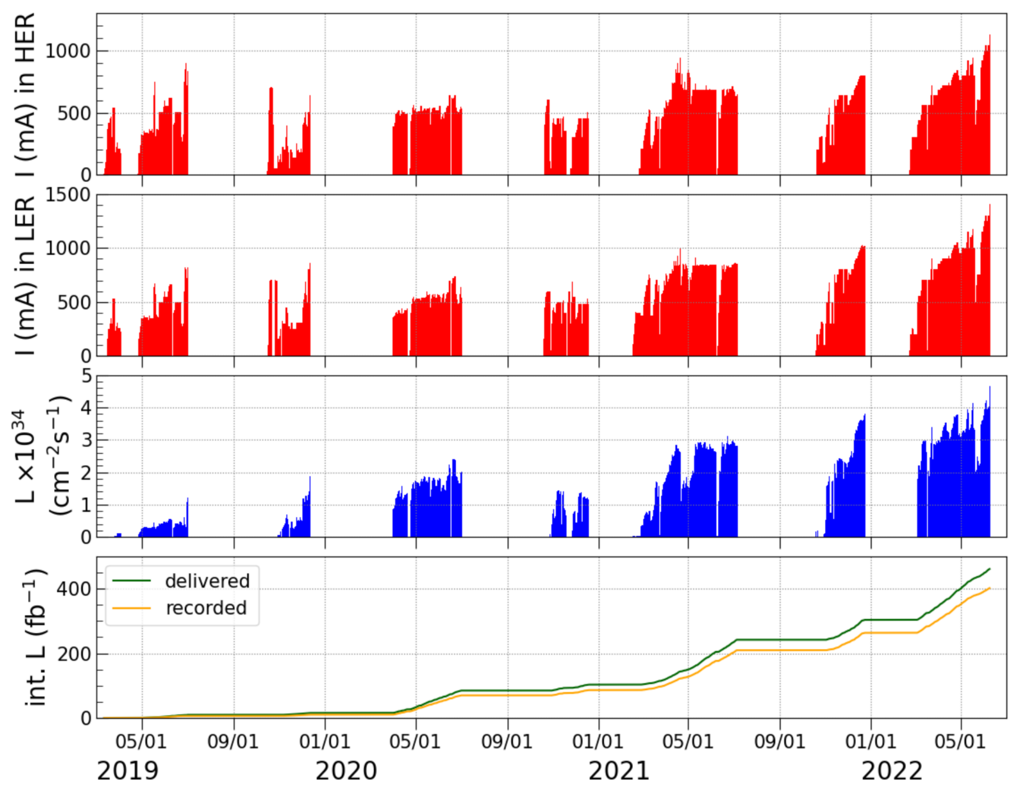


Figure 1.3: Luminosity history of the SuperKEKB collider [2].

- Particle identification system (PID): Identifies particles in the barrel and end–cap region. It is comprised of two components, a time of propagation (TOP) and an aerogel ring imaging Čerenkov (ARICH) detector.
- Electromagnetic Calorimeter (ECL): Detects photons and electrons over a wide energy range.
- K_L Muon Detector (KLM): As the name suggest, the main purpose of this subdetector is to identify K_L and muons. It is made out of alternating layers of iron plates and detector components located outside the superconducting solenoid.

All in all, the detector as a whole should meet the following requirements, as describe in [36]:

- Immaculate vertex resolution in the order of $10^{-4}m$;
- high reconstruction efficiencies for photons and charged particles;
- very good momentum resolution and precise measurements of photon energy and direction;
- high particle identification efficiencies, so that pions can be separated from kaons;

- coverage of most of the solid angle of the detector;
- an agile trigger system;
- strong capability to record large amounts of data in a short amount of time.

Finally, due to the vast amount of data, a large distributed computing grid is required to analyse the collected data.

1.3 Monte Carlo Simulations in High Energy Physics

Analyzing data collected from a detector is not trivial. Precise preparation has to be done beforehand. Performing any type of measurements on this data requires the modeling of the experiment as good as possible to first prepare each analysis. In the case of most experiments in high energy physics and the Belle II experiment this is done in the form of Monte Carlo simulations. It allows fine tuning physics analyses to be able to filter the signal, the primary physics process being searched for, from everything else, the background. Since a large portion of the measurements will be focused on so-called rare processes, these are processes with a branching fraction of 10^{-6} and lower, a strong statistical knowledge of the backgrounds is required to accurately distinguish the signal from the remaining background. The approach taken during the Belle experiment was to simulate ten times the volume of measured data. To use the same approach at Belle II would then require simulations of an integrated luminosity of 500 ab^{-1}. Data simulated through MC simulations pass trough the following stages:

- Generate: In the generation stage, the physics event itself is created. Here, hadronic decays and its corresponding quark hadronisation are simulated. The collision events of the electron positron pairs are simulated including all decays into the final state particles using the EvtGen [33] package and Pythia [46].
- Simulate: In the simulation stage, the detector is simulated. Interactions of the decays generated in the last stage with the Belle II detector are calculated using the *Geant*4 toolkit [6]. The result of this stage looks as close to the data that is collected in the Belle II experiment as possible, including track candidates and particle trajectories etc. However, in comparison to regular data taking, the MC data also includes the information of the MC simulations, like the true PIDs.
- Reconstruct: In the reconstruction stage, decayed particles and other useful variables are reconstructed from the simulation stage. This mimics the process for experimentally collected data with the actual detector.
- Skim: In the skim stage, the data is reduced to remove events that are not relevant for a given physics analysis. This stage is study specific and only selects suitable events for a given signal. This study makes use of the Full Event Interpretation (FEI, see chapter 1.6) [29] to filter appropriate events.

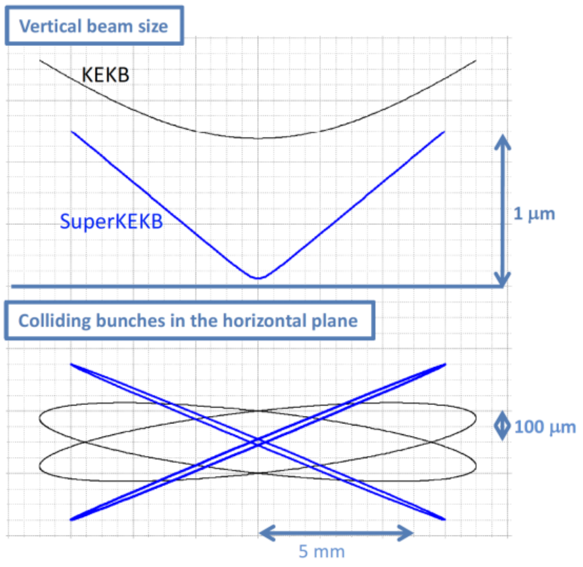


Figure 1.4: Schematic overview of the nano beam scheme.

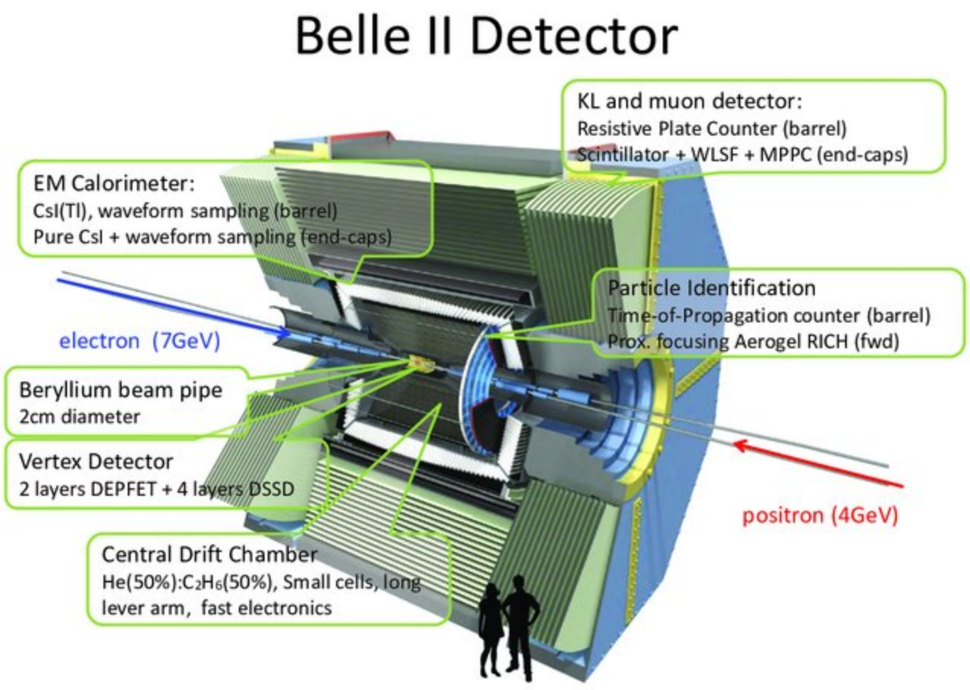
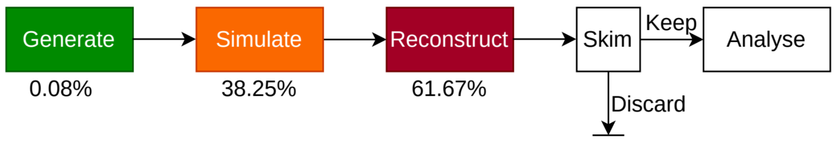
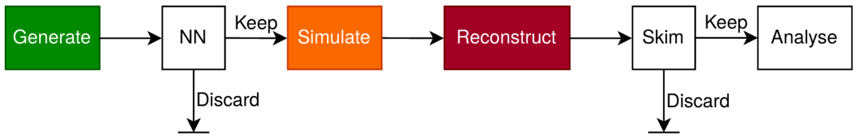

Belle II Detector
KL and muon detector:
Resistive Plate Counter (barrel)
Scintillator + WLSF + MPPC (end-caps)
EM Calorimeter:
CsI(Tl), waveform sampling (barrel)
Pure CsI + waveform sampling (end-caps)
electron (7GeV)
Particle Identification
Time-of-Propagation counter (barrel)
Prox. focusing Aerogel RICH (fwd)
Beryllium beam pipe
2cm diameter
Vertex Detector
2 layers DEPFET + 4 layers DSSD
positron (4GeV)
Central Drift Chamber
He(50%):C2H6(50%), Small cells, long lever arm, fast electronics

Figure 1.5: A cross section drawing of the Belle II detector.



(a) Regular Monte Carlo simulations.



(b) Smart Monte Carlo simulations.

Figure 1.6: Regular and smart Monte Carlo simulations within the Belle II experiment [27].

- Analyse: The analysis stage is the actual physics analysis, where the real experimental data is compared to the MC simulations in the quest to find new physics.

The issue with this conventional way of generating MC events is that this approach is extremely inefficient. During the skim, most events are going to be trivially discarded as irrelevant to the particular study. In addition, a brute force simulation of all the data is infeasible using current or near-future computational resources. There are two immediately and obvious solutions to solve this problem: Either only simulating events that are useful for a given physics analysis, or increase the speed of simulating events, thus removing entirely the existing bottleneck. While the second solution is undoubtedly the more beautiful solution, it is extremely difficult, if not impossible, to implement. The sheer complexity of simulating every single interaction with every component of the Belle II detector is too much. In contrast, the first solution where only events that are relevant to a particular study are simulated is feasible. This study follows the idea pioneered by Kahn [27]. In his work a neural network is injected between the event generation and the simulation stage of the MC simulations as shown in figure 1.6b. The network is trained to identify the relevant events from the information that is available at the event generation stage, thus in the optimal case, exclusively events are discarded which would have been discarded by the Skim stage in any case.

1.4 The Belle II Analysis Software Framework

The Belle II Analysis Software Framework (basf2) [32] is the software package behind the Belle II experiment. It is used for data recording, simulation, reconstruction and analysis/fitting. Everything that is needed for the process shown in figure 1.6a is included in basf2. The package itself is written in $C{+}{+}$, but it also supports custom, user-developed Python modules. An integration of machine learning libraries is thus also possible, allowing the speedup of the MC simulation process with neural networks.

1.5 The Tagging Method

In many studies, physicists have to deal with missing energies in the final states. This energy difference usually occurs in events containing neutrinos that can not be detected. The possibility to fully reconstruct an $\Upsilon(4S)$ decay is crucial for any analysis of processes with missing energy. To measure the branching fractions of processes such as $B \to K^{(*)}\nu\nu$ or $B \to l\nu\gamma$, the signal from the electron positron collisions must be identified for which the tag side has to be fully reconstructed to infer the signal side. In the scope of this study, MC simulations are used to preprocess the analysis to accurately distinguish the signal from the background. During the reconstruction, signal events are selected trough the so called tagging method as seen in figure 1.7 and is a 4 step process:

1. Reconstruct generically decaying B mesons, the so called B_{tag}, with the help of the FEI (chapter 1.6).
2. Reconstruct the signal side of the decay, the B_{sig}, with the help of experimental data.
3. Pair up the two mesons from above to form an $\Upsilon(4S)$ resonance.
4. Send the reconstructed $\Upsilon(4S)$ resonance to the skimming stage to filter out irrelevant events, while retaining as many signal events as possible.

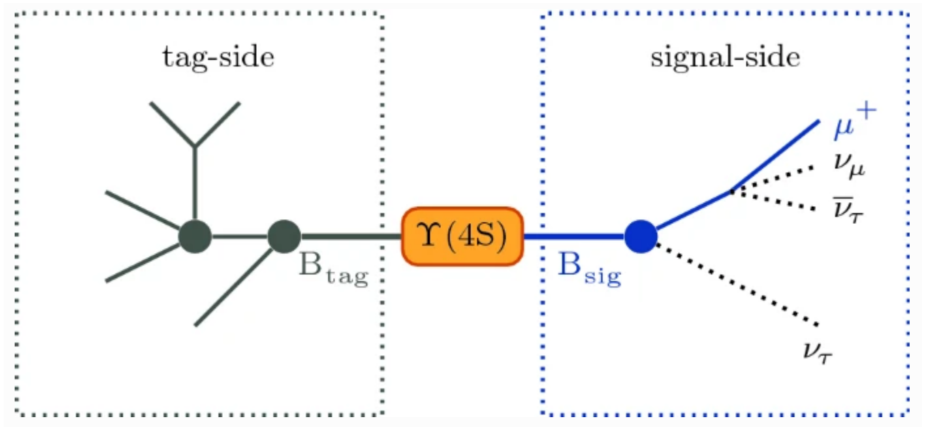


Figure 1.7: B tagging at Belle II.

1.6 The Full Event Interpretation

The full event interpretation (FEI) is the tool used during the tagging method at Belle II to reconstruct generically decaying events [29]. It is an exclusive tagging algorithm which enables precise measurements of otherwise inaccessible B decays. With the help of a variety of modern machine learning methods, the FEI can automatically identify plausible

B meson decays. With the use of experimental data from the detector, the FEI yields greater performance and efficiencies to provide a large effective sample size.

The general procedure of the FEI is shown in figure 1.8. It is based on a hierarchical structure using reconstructed tracks and clusters as input. In the next step, using the input data, the FEI reconstructs the possible final state particles. Next, the FEI translates the final state particles to intermediate particles to finally reconstruct a B meson. With the help of multiple fast Boosted Decision Trees (fBDT) the FEI is able to provide its confidence level of a reconstructed particle candidate, based on its own properties, but also its parents status. In conclusion, the output of the FEI is an array of reconstructed B mesons for each input with the corresponding confidence levels, the so called signal probabilities (sigProb) of the reconstruction, indicating the probability of the reconstruction being correct.

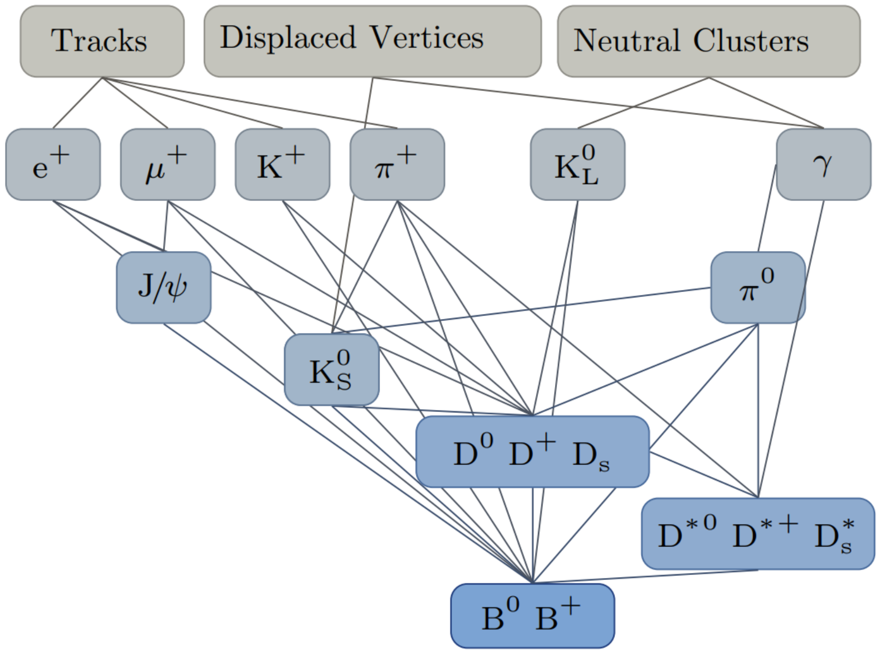
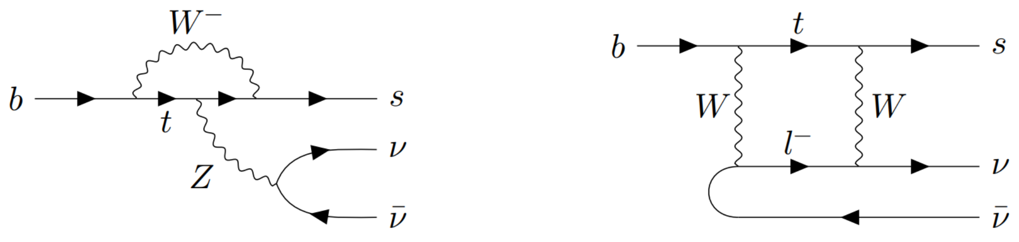


Figure 1.8: Overview of the hierarchical structure of the FEI.

Chapter 2

Theoretical Framework

The goal of this thesis is to investigate the performance of the smart background Monte Carlo simulations on a specific analysis. For this, the decay that was chosen is the set of process that lead to the decay $B^0 \to K^{(*)}\nu\nu$. This process is part of a subset class known as flavor changing neutral currents (FCNCs). It involves the transition of a similarly charged quark to a different flavor, e.g. $b \to s$ or $c \to u$. This is a process that is not allowed on tree level in the standard model. This is because neutral currents in the standard model can only be mediated by the Z and γ bosons in first order, which do not alter the flavor. Instead the transition can only occur in higher order processes, involving multiple quark flavor changes. The famous penguin and box diagrams in figure 2.1 show the leading order contributions to the $B \to K^{(*)}\nu\nu$ amplitudes with respect to the standard model. This process is extremely suppressed, so any sizable branching fraction would point to new physics, making them a perfect candidate to probe the Standard Model.



(a) Penguin diagram. (b) Box diagram.

Figure 2.1: Feynman diagrams for dominant contributions to the $b \to s\nu\bar{\nu}$ transition in the Standard Model.

1. Generation	2. Generation
$\begin{pmatrix}\nu_e \\ e^-\end{pmatrix}_L$, $(\nu)_R$ and $(e^-)_R$	...
$\begin{pmatrix}u \\ d\end{pmatrix}_L$, $(u)_R$ and $(d)_R$	...

Table 2.1: Eigenstates of the Standard Model.

2.1 Flavor Changing in the Standard Model

Quark flavor changing between different generations is described by the Yukawa Lagrangian,

$$\mathcal{L}_{\mathrm{Y}} = -Y_u^{ij}\bar{q}'_{iL}\tilde{\phi}u'_{jR} - Y_d^{ij}\bar{q}'_{iL}\phi d'_{jR} - Y_\ell^{ij}\bar{\ell}'_{iL}\phi e'_{jR} + \text{ h.c } , \tag{2.1}$$

where the quarks and leptons couple to the Higgs field. The Yukawa Lagrangian containing flavor–changing interactions and can be written in terms of quark and lepton interactions as shown in equation 2.1. All $Y_{u,d,l}^{ij}$ are Yukawa couplings (3×3 matrices) between fermion families i and j. q'_L denotes the weak eigenstates, ϕ the Higgs doublet and l'_L and e'_R are left handed lepton doublets and right handed charged lepton singlets as shown in table 2.1.

All in all, there are two bases that can be used to describe quarks:

1. The mass basis, where the Yukawa terms (mass terms) are diagonal and correspond to the physical particles.
2. The weak basis, where the so called weak eigenstates are diagonal with respect to this basis. The weak eigenstates are defined as: $\begin{pmatrix}u \\ d'\end{pmatrix}_L$ where $d' = d \times cos\phi_c + s \times sin\phi_c$ in the first two generations. ϕ_c represents the mixing angle and is usually referred to the Cabibbo-angle.

The simple fact that these two basis are not the same results in flavor changing in the Standard Model. In three generations, the mixing angle and hence the strength of the flavor changing is described by the Cabibbo-Kobayashi-Maskawa (CKM) matrix. In other words, because the V_{CKM} matrix is not diagonal, the W bosons are able to couple to quark mass eigenstates of different generations. It is important to note that in the Standard Model, the V_{CKM} matrix is the only source of flavor changing, thus this should describe the full extent of flavor changing. The unitary V_{CKM} matrix can be written as follows:

$$V_{\mathrm{CKM}} = \begin{bmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{bmatrix} \tag{2.2}$$

where the mass generations are sorted by their mass. With the Wolfenstein parameterization [11], the matrix can be expressed with four free parameters; three real and one imaginary. Taylor expanded with respect to λ, V_{CKM} can be expressed as:

$$V_{\mathrm{CKM}} = \begin{bmatrix} 1-\lambda^2/2 & \lambda & A\lambda^3(\rho - i\eta) \\ -\lambda & 1-\lambda^2/2 & A\lambda^2 \\ A\lambda^3(1-\rho-i\eta) & -A\lambda^2 & 1 \end{bmatrix} + \mathcal{O}\left(\lambda^4\right) \tag{2.3}$$

Current measurements have shown the following magnitudes for the matrix elements [51]:

$$V_{\mathrm{CKM}} = \begin{pmatrix} 0.97446 \pm 0.00010 & 0.22452 \pm 0.00044 & 0.00365 \pm 0.00012 \\ 0.22438 \pm 0.00044 & 0.97359^{+0.00010}_{-0.00011} & 0.04214 \pm 0.00076 \\ 0.00896^{+0.00024}_{-0.00023} & 0.04133 \pm 0.00074 & 0.999105 \pm 0.000032 \end{pmatrix} \tag{2.4}$$

2.2 Flavor-Changing Neutral Currents

Due to the fact that only the W boson is able to change quark flavor, flavor-changing neutral currents can exclusively occur at one loop level or higher. All flavor changing on tree level must have a non neutral current. The Glashow-Iliopoulos-Maiani (GIM) mechanism [35] describes the strength of interaction for processes on loop-level. The GIM mechanism takes advantage of the fact that the V_{CKM} matrix is unitary and thus any pair of rows or columns are orthogonal. In the penguin diagram shown in figure 2.1, the CKM dependence is fully described through

$$\mathrm{M} \propto \sum_i V_{ib}^* V_{is} = 0, \tag{2.5}$$

which vanishes due to the orthogonality property. For flavor changing on loop-level, the masses of the quarks themselves play an important role. They can be used to change the basis and produce off-diagonal couplings. However, for this universal matrices have to be used, which are matrices proportional to the identity. The mass matrices m_i are not universal, which means they have to be projected onto a different basis through the Inami–Lim function [25]:

$$\mathrm{M} \propto \sum_i V_{ib}^* V_{is} f(m_i), \tag{2.6}$$

where any terms independent of m_i must vanish (can be rotated away). The Inami–Lim function introduces two interesting properties into flavor changing neutral currents:

1. Heavier intermediate quark masses are expected to dominate the loop contributions, hence the transition is expected to be most sensitive to the top quark.

2. The closeness of quark masses of different generations directly suppresses the amplitude.

This underlines an important property of high precision experiments like Belle II. Despite operating at energies lower than the mass of the top quark, the measurement of its off–shell

Mode	$\mathcal{B}_{SM}$ [10^{-6}]	$\mathcal{B}_{\text{exp}}$ [10^{-6}] (90% C.L.)
$B^+ \to K^+ \nu\bar{\nu}$	4.68 ± 0.64	< 16(BaBar 2013)
$B^0 \to K^0_S \nu\bar{\nu}$	2.17 ± 0.30	< 13 (Belle 2017)
$B^+ \to K^{*+} \nu\bar{\nu}$	10.22 ± 1.19	< 40 (Belle 2013)
$B^0 \to K^{*0} \nu\bar{\nu}$	9.48 ± 1.10	< 18 (Belle 2017)

Table 2.2: Standard Model prediction and previous measurements for a variety of $B \to K\nu\bar{\nu}$ signal channels [15].

effects on higher order processes allows an indirect examination of its properties. In addition, it is possible to probe for any other heavy particles involved, including Higgs or new physics particles.

Lastly, due to the ability to handle extremely high luminosity, Belle II is very well equipped to analyze these FCNCs and the process $B^0 \to K^{(*)}\nu\nu$ is an extremely good candidate to probe the standard model in the search for NP.

Chapter 3

Machine Learning

Machine learning is part of the broader topic of artificial intelligence (AI). AI describes the ability to build software applications to represent an intelligent behavior in the scope of a specific problem set. In addition, machine learning, which is part of AI, enables the capability of machines to learn from data, and to make predictions about new data. The goal of this chapter is to introduce the basics of machine learning and give detail about the specific machine learning algorithms used within the scope of this thesis. This chapter will closely follow some sections of the excellent book by Ian Goodfellow [23].

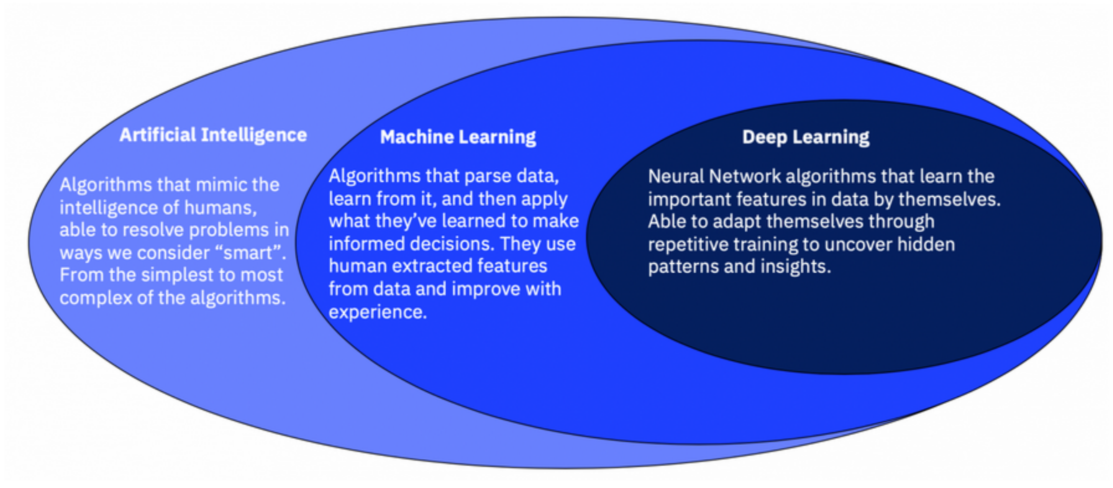

Artificial Intelligence
Algorithms that mimic the intelligence of humans, able to resolve problems in ways we consider "smart". From the simplest to most complex of the algorithms.
Machine Learning
Algorithms that parse data, learn from it, and then apply what they've learned to make informed decisions. They use human extracted features from data and improve with experience.
Deep Learning
Neural Network algorithms that learn the important features in data by themselves. Able to adapt themselves through repetitive training to uncover hidden patterns and insights.

Figure 3.1: Relations of Artificial Intelligence, Machine Learning and Deep Learning [20].

3.1 Types of Machine Learning

Machine learning itself is a vast and well researched area with many subcategories. It is part of the broader topic of artificial intelligence as shown in figure 3.1. A great summary all these subcategories is given in [20]. In general machine learning can be split into several areas, which include:

- **Supervised**: Supervised machine learning algorithms require a large amount and labeled training data set. It is most commonly used for predictive modeling. During training, the algorithm can compare its prediction for given input data to the correct output and modify the model accordingly. This subcategory is used for the smart background Monte Carlo filter.

- **Unsupervised**: Unsupervised machine learning algorithms do not require label training data. Instead, the algorithm does not try to predict a certain outcome, but rather aims to identify the underlying structure of the data. For example, unsupervised machine learning algorithms can cluster data into groups based on their similarities.

- **Semi–supervised**: Semi-supervised machine learning algorithms is a mix between unsupervised and supervised learning. Typically a small labeled training data set is used to optimize an unsupervised algorithm, which is then applied to a large unlabeled data set. The main use case is to detect anomalies in data, which require some background information to train.

- **Reinforcement**: Reinforcement machine learning is a sort of behavioral learning model. It has some similarities to supervised learning where data with the corresponding output is used. However, instead of using sample data, this model learns as it goes by using trial and error. By doing this, the alrogithm sort of creates the training data itself. Currently, the most well known application is learning games such as go and chess [45], however in recent years reinforcement learning has also been applied in the real world [41].

The topic of machine learning is not new, with first advances made in the 1950s [21]. However, only within the past few decades have computational resources progressed enough to make deep learning feasible. Today machine learning is used in almost all areas of physics, including high energy physics [16]. Within the scope of this thesis, exclusively supervised learning algorithms are applied, which I will go into more detail in the next section.

3.2 Supervised Machine Learning Basics

A formal description of what it means to learn from data is given by Mitchell et. al. [37]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

There can be a large variety of experiences E, tasks T, and performance measures P which I will provide an intuitive description and tie it to the smart background Monte Carlo simulations in the next sections.

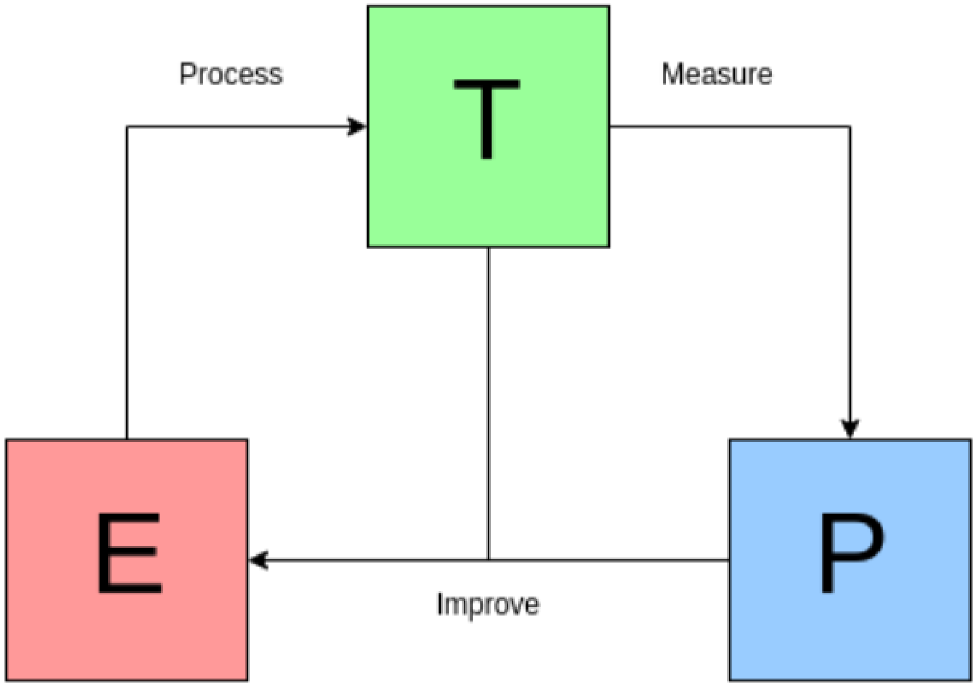


Figure 3.2: The Mitchell Paradigm [37].

3.2.1 The Task T

In general, machine learning enables us to write scripts that solve tasks which are too difficult to work out by hard coding a solution. These are usually tasks designed for humans rather than machines like image recognition. These problems entail an underlying understanding of data rather than simple rule based decisions. In this very formal definition of the task T, the process of learning is not the task, but learning is a means to be able to tackle the task. Machine learning tasks are described in terms of how machine learning systems should process an example.

In the case of smart background Monte Carlo simulation, the task at hand is a classification problem with an adjusted performance measure, which is explained in detail in section 5.4. The computer program is asked to specify which categories k (event will or will not pass the skim) the data belongs to. Here the algorithm learns a function

$$f : \mathbf{R}^n \rightarrow 1, \ldots, k. \tag{3.1}$$

Mitchell Paradigm	Smart Background Simulation
Task T	Classification Problem True if event will pass the skim False if event will fail the skim
Experience E	Data: Graphs of Events Label: Pass or Fail
Performance P	Speedup metric

Table 3.1: Machine Learning System used compared to the Mitchell Paradigm

When $y = f(\mathbf{x})$, the model assigns a class y to an input described by vector $\mathbf{x}$, which is the information available after the event generation stage.

3.2.2 The Performance Measure P

The performance measure P is a function that takes the output of the algorithm as input and returns a scalar value. The performance measure is used to measure the quality of the output of the algorithm, but can also be used to determine if the algorithm has learned the task. Usually this performance measure P is specific for each task T. At first, the choice of the measures may seem simple and objective, but in many cases this is not as straight forward as it sounds. The most common measure used for classification is the accuracy of the model. Accuracy is just the proportion of examples for which the model produces the correct output. However, in the case of the smart background Monte Carlo simulations, the accuracy itself is not directly of interest, instead the speedup should be maximized. This measure is explained in detail in section 5.4. In general, we are interested in how well our models preform on data that it has not seen before, since this determines how well it will work when deployed in the real world. Thus the evaluation of P is done using a test set, which is separate from the data used for training the machine learning model, the train set.

3.2.3 The Experience E

The experience E describes an entire dataset used for training and evaluating a model. A dataset is a collection of many examples, described as $\mathbf{x}$ in section 3.2.1. Supervised learning algorithms experience a dataset containing features and its associated labels. In essence, for the smart background Monte Carlo simulations, the features $\mathbf{x}$ represent the data available after event generation, and the label y represents if an event is going to pass the skim or not.

All in all, the machine learning model used in the smart background Monte Carlo simulation is summarized in table 3.1

3.3 Training Supervised Learning Algorithms

The definition of a machine learning algorithm using the Mitchell Paradigm, that is an algorithm capable of improving a computer program's performance at some task given some experience, is very abstract. To make this more clear, it is important to understand how exactly the algorithm improves its performance during the training process. In general, a models output is compared to the true target with regards to a certain metric. This value is then used to update the models parameters in a certain way that optimizes this value. This metric is the performance P of the model and is commonly referred to as the loss function. During training, one tries to minimize the loss. After one comparison, the program should determine how exactly to adjust the internal parameters in order to increase the performance and get better at the task. Next, the model gives it another try and repeats the process. The update rule is referred to as the optimizer. Even though there exist a large variety of optimizers, most of them are gradient based. In a step called backpropagation the gradient, which is the derivative of the loss function with respect to the model's weights, is efficiently calculated. In addition, the update of the model itself is completed using this metric. Each update is called a step and all steps over the complete training dataset is called an epoch. Due to limited computational resources, the maximal number of epochs is restricted. This restriction should be chosen in a way, so that the magnitude of the loss in the last epoch will not decrease, even if we would continue to train the model. In other words, the model should be in its optimal state and no improvements should be possible, before training is aborted. In addition, it is very important to consider the size of each update taken at every step, which is referred to as the learning rate. Lower learning rate means that the training will consume more computational resources, while a higher rate increases the risk of not finding the optimal solution as shown in figure 3.3.

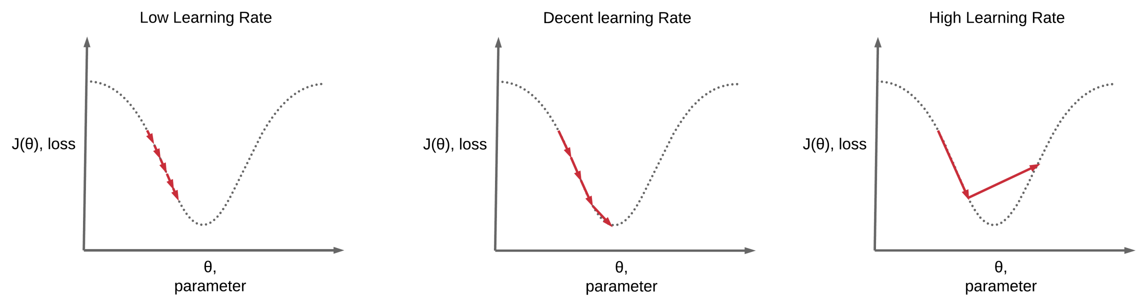


Figure 3.3: Choosing the correct learning rate [42].

This gradient based optimization problem is carried out over one set of training data. To measure how well the ML model generalizes in the real world, the performance is evaluated on new and previously unseen data that is independent from the training itself, which is defined as the test error. It is important to keep track of the train and test errors and compare them after each epoch. Divergence between these two values indicates that the model is not in its optimal state. One differentiates between underfitting and overfitting:

- Underfitting: The model is not able to learn the underlying structure of the training data. This will result in a high train error. This often means that the model itself is chosen in a way, that it can not represent the structure of the dataset, as there are not enough trainable parameters.
- Overfitting: The model is overtrained. Instead of learning the underlying structure of the data, the model tries to fit perfectly to all datapoints within the training dataset. Even though this minimizes the training loss, the test loss will diverge from the train loss, resulting into a bad performance when applying the model in the real world.

The tendnancy of a model to overfit is measured by the capacity, which is a measure of the ability of a model to memorize the detailed properties of the training set. In general, a more complicated model with a high amount of learnable parameters will have a higher capacity, reducing the loss on the training set, but increasing the risk of overfitting as seen in figure 3.4.

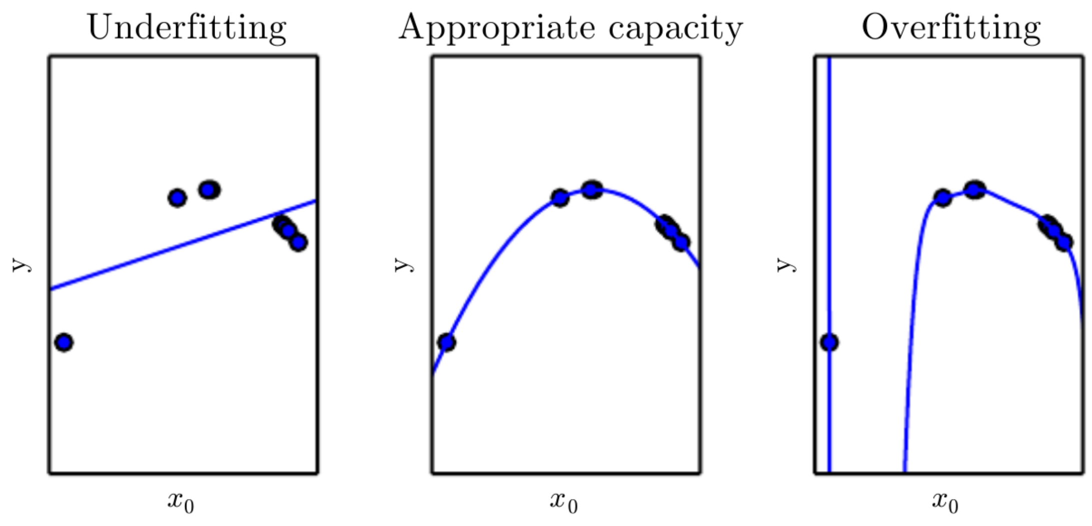


Figure 3.4: Three different fits to the same dataset. The first image (left) shows a linear fit to a polynomial dataset, which yields an underfitted model. The model is not complex enough to efficiently learn the underlying structure of the data. The second image (center) shows a polynomial fit to a polynomial dataset, return an accurate fit. Neither overfitting, nor underfitting is occurring and this model can be generalized to new unseen data very well. Lastly, the third image (right) represents a model with too many learnable parameters. This model is overfitted and cannot be used to generalize to new unseen data.

Another more formal description of over–and underfitting is given by the Bias and Variance Tradeoff. Bias is defined as the difference between the expectation value of the estimated result by the ML algorithm and the true value:

$$\text{Bias}\left(\hat{\theta}_{\text{model}}\right) = \mathbb{E}\left(\hat{\theta}_{\text{model}}\right) - \theta_{\text{true}} \tag{3.2}$$

The variance is the standard variance defined as

$$\text{Variance}\left(\hat{\theta}_{\text{model}}\right) = \mathbb{E}[(\hat{\theta}_{\text{model}} - \mu)^2]. \tag{3.3}$$

Variance and bias represent two distinct sources of error. While the bias is the expected deviation from the real function, the variance refers to the change in the model when using different portions of the training dataset. Usually with decreasing variance, the bias increase and vice versa. Figure 3.5 shows this tradeoff. The optimization problem boils down to find the optimal mix of variance and bias to minimizes the loss and optimize the capacity.

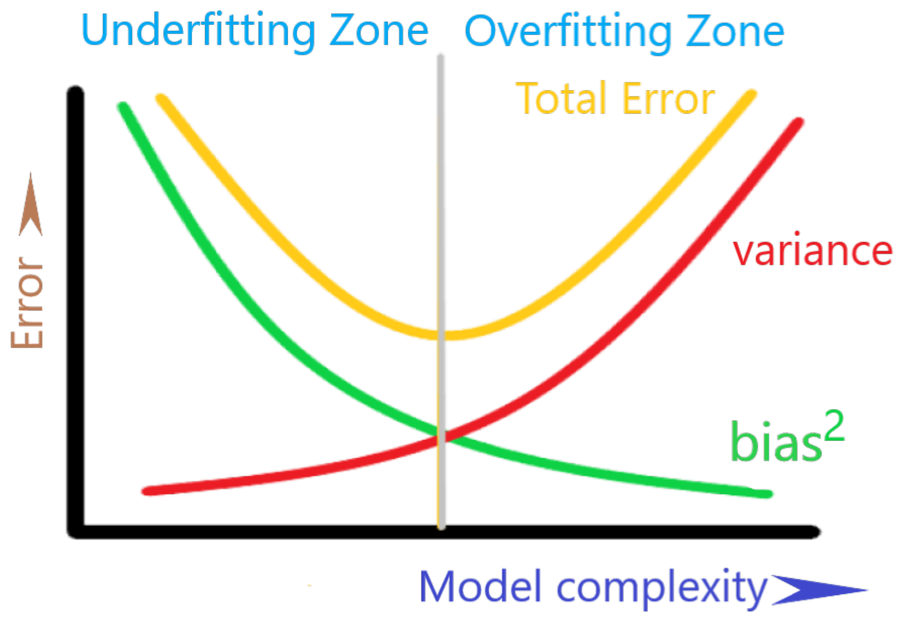


Figure 3.5: The Variance Bias Tradeoff. Increased variance leads to decreasing bias and vice versa.

3.4 Supervised Learning Algorithms

Learning theory suggests that ML algorithms can generalize well from a finite training set. However, inductive reasoning or inferring rules form limited data is generally not possible. To logically infer a rule describing every member x from a set A, one must have information about every member $x \in A$. The way machine learning algorithms avoid this pitfall is to only offer probabilistic rules, rather than certain outcomes. ML provide rules that are probably correct for most members of the set. Unfortunately, this does not solve the entire problem. The **no free lunch theorem** [50] states that, averaged over all possible data distributions, every classification algorithm yields the same performance on previously unobserved points. In other words, there is no one magical ML algorithm that universally generalizes to all data. In some sense, even the most sophisticated algorithms perform the same as random guessing. Luckily, this only holds for averages over all tasks, but not for individual tasks. This means that the goal of ML research is not to look for a universal algorithm, but to develop models that perform extremely well on one specific task, which is why the exact choice of model is important.

Within the topic of supervised machine learning, a whole range of different algorithms exists. These include, but are not limited to:

- **Linear Regressions:** The class of linear regression is intended for target values which are expected to be a linear combination of features. The most common model used is a fit using Ordinary Least Squares.
- **Gaussian Processes (GP):** GP include supervised learning methods designed to solve regression and probabilistic classification problems. For example, the Gaussian Process Regressor implements Gaussian processes (GP) for regression purposes.
- **Support Vector Machines:** SVMs are a set of learning algorithms used for classification, regression and outliers detection which are especially effective in high dimensional space.
- **Decision Trees:** Decision trees are a set of algorithms that are non–parametic. The goal is to create a model that predicts the value of a target by learning simple decision rules inferred from the data features.
- **Ensemble methods:** The goal of ensemble methods is to combine the predictions of several base estimators to create a better prediction.
- **Neural Networks:** Neural networks are a set of algorithms that are based on the idea of a network of neurons. Trainable parameters can be increased almost indefinite making them a great choice for extremely complicated tasks.
- **Etc.**

For this research project, decision trees are used as part of the FEI 1.6, while neural networks are utilized for the smart background filter due to the underlying structure of the data. These two algorithms will be explained in detail in the next sections.

3.5 Neural Networks and Deep Learning

Neural networks and deep learning are among others algorithms within supervised learning, which are mostly used for extremely complicated tasks. These computing systems are inspired by information processing that can be found in biology, with nodes (neurons) and edges (synapses) that connect them. Information is then propagated by the nodes through their corresponding edges, layer by layer. The term deep learning then comes from introducing a whole range of layers on top of each other, making the network ”deep”.

The universal approximation theorem [24] has shown that a combination of any non–linear functions can approximate any other function. Thus by including non–linear activation functions in the nodes, the network can be used to learn complex functions. Because of this, image recognition and other extremely complicated tasks are preformed with neural networks.

3.5.1 Perceptrons and Artificial Neurons

The building blocks of NNs are artificial neurons. The simplest type of these neurons is called a perceptron. Several binary inputs $x_1, \ldots, x_n \in [0, 1]$ are fed into a perceptron to produce a single binary output as shown in figure 3.6. Different weights, $w_1, \ldots, w_n$ of real numbers expressing the importance of the respective inputs to the output can be introduced. The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_{j=1}^{n} w_j x_j$ is less than or greater than some threshold value. Similar to the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

$$\text{output} \ = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \ \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \ \text{threshold} \end{cases} \tag{3.4}$$

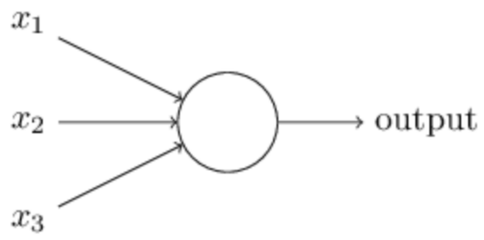


Figure 3.6: Schematic drawing of a perceptron with three inputs.

In some sense, a perceptron is a device that makes decisions by weighing up evidence. However, a single perceptron is very limited in the amount of data it can efficiently handle to approximate more complex functions. Luckily it is possible to combine many of the machines to create a more powerful machine as just like the model in figure 3.7. Here perceptrons in the second layer take the output of the perceptrons in the first layer as input, increasing the complexity of the system. In this way, a many–layer network of perceptrons can engage in sophisticated decision making. To simplify equation 3.4 we can make two notational changes. Firstly the inner sums can be rewritten as dot products $w \cdot x := \sum_{j=1}^{n} w_j x_j$. The second change is to shift the threshold to the left side of the equation by introducing a bias term b:

$$\text{output} \ = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{3.5}$$

The bias can be thought as a measure of how simple it is to get the perceptron to output a 1, where a higher bias means it is more likely to get a 0 as output.

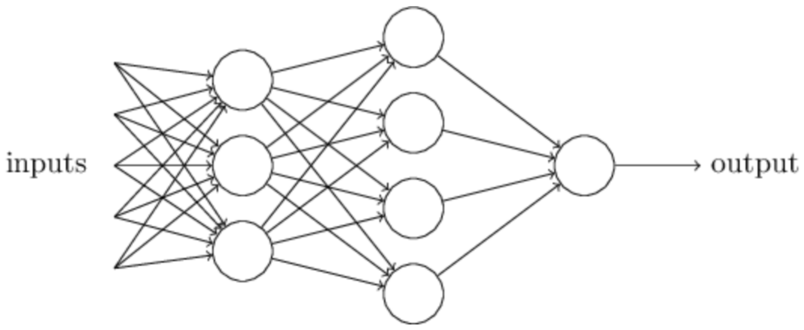


Figure 3.7: Combination of many perceptrons producing a large, more complex approximator.

As mentioned before, a perceptron is a special type of neuron that takes a step function as activation function and only allows binary inputs. The major flaw with these simple perceptrons is that training them is extremely difficult [38]. Specifically, a small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip, say from 0 to 1. This drastic shift may cause the behavior of the rest of the network to completely change in some very complicated way. That makes it difficult to see how to gradually modify the weights and biases so that the network gets closer to the desired behavior. To combat this problem, in modern feed forward networks a more general type of neurons are utilized introducing the ability to approximate even more complex functions. A very common neuron is the so called sigmoid neuron. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output. That's the crucial fact which will allow a network of sigmoid neurons to learn. The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \text{ where } x = \sum_{j=1}^{n} w_j x_j + b. \tag{3.6}$$

The similarity to the perceptron model lies in the fact that $\sigma(z) \approx 1$ for large z and $\sigma(z) \approx 0$ for small z, mimicking the same behavior in the limit as shown in figure 3.8. In reality, a whole range of different (non–linear) activation functions can be used.

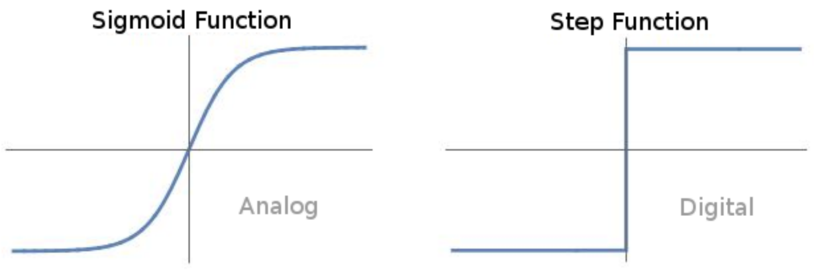


Figure 3.8: Sigmoid function (left) compared to the perceptron model (right). While the step function is binary and behaves like a digital machine, the sigmoid function is differentiable and yield a continuous and analog function.

3.5.2 Graph Neural Networks

To work with graph structured data, so called graph neural networks (GNNs) [43] are used. To understand these types of models, we first need to define a graph. In the most general case, graphs are simply a set of objects (nodes) and their corresponding interactions (edges). For example, to represent a social network as a graph we might use nodes to represent individuals and use edges to represent that two individuals are friends. In addition, we could also have a look at a decay of a heavy particle where nodes are the particles with their properties and the edges represent the mother daughter relationships. At the Belle II experiment, an example of such a tree structured graph is shown in figure 3.9. The power in graph structured data lies in the fact that it includes the relationship between points.

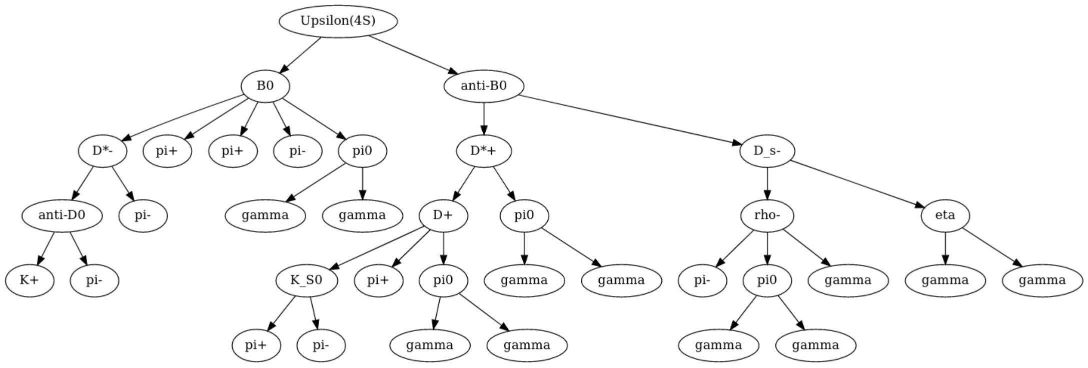


Figure 3.9: Example of a standard Belle II decay represented as a graph structure with particles as nodes and mother and daughter relationships as edges.

Formally, a graph $G = (V, E)$ is a set of nodes V and a set of edges E between the nodes. Edges going from node $u \in V$ to node $v \in V$ as $(u, v) \in E$. In this thesis we only consider semi–simple undirected graphs, meaning there is at most one edge between any two nodes and all edges are undirected, i.e. $(u, v) \in E \leftrightarrow (v, u) \in E$. However, in contrast to regular simple graphs, the nodes are allowed to have self loops. In fact, every node will have an edge with itself. To illustrate this, figure 3.10 shows an example of a directed graph with the following properties:

- 5 nodes: $1, 2, \ldots, 5$.
- 5 edges: $(1, 2), (1, 4) \ldots (2, 1)$ represented by letters.
- $5 + 5 = 10$ labels. Each edge and node have a corresponding label.

It is important to note that in this context, labels are not the same as the labels of the labeled dataset. Instead they are used to represent the features within a graph. In literature they are also often referred to as node and edge features.

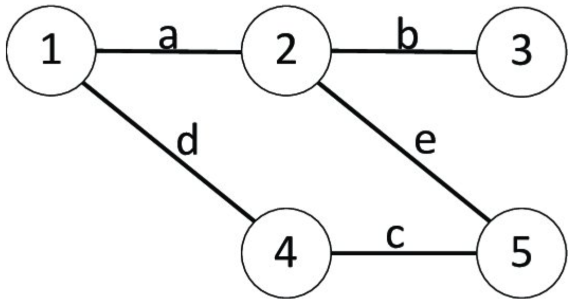


Figure 3.10: An example of an undirected graph with $n = 5$ and $v = 5$ [44].

Now that we have a formal definition of a graph, it is important to consider the training process. For supervised training on graphs, a learning framework can be formalized as follows:

$$\begin{aligned} \mathcal{L} = \{(G_i, F_i, t_i) \mid G_i = (N_i, E_i)\,; F_i = F_{N_i} \in \mathbb{R}^m \\ t_i \in \{0, 1\}, 0 \leq i < p\}\,. \end{aligned} \tag{3.7}$$

Here G_i is one of p graphs with nodes N_i and edges E_i. F_i is the feature vector with m dimensions, which are only attributed to the nodes. Lastly, t_i is the target or label of its corresponding graph at position i. Graphs do not have to be fully connected, meaning that some regions of nodes may not be connected to each other. In other words, all the graphs inside the learning set can be combined into a single disconnected graph G. Therefore the framework can be summarized as:

$$\mathcal{L} = \big\{(G, F, T) \mid G = (N, E); F = \{F_{N_i}\} \in \mathbb{R}^{m \times p}; t \in \{0, 1\}^p\big\} \tag{3.8}$$

To efficiently make use of GNNs, it is important to specify a method to update the network. In some sense, what is the best way to approximate a function that can map a graph G to a target T. The simplest form of a GNN is a graph convolutions network (GCN) [52].

The mathematical expression of such a network can be formalized as follows:

$$h_i^{(l+1)} = \sigma(b^{(l)} + \sum_{j \in N_i} \frac{h_j^{(l)}}{c_{ij}} W^{(l)}) \tag{3.9}$$

where N_i represents all the neighboring nodes of the node i, h_i^l expresses the feature vector of node i in the l–th step, $W^{(l)}$ is the corresponding weight matrix, c_{ji} is the normalization factor, σ is the activation function and b^l the bias vector. The complete network gets upgraded through convolutions at every node with its immediate neighbors.

3.5.3 Attention Mechanism

In literature, many authors have shown that an attention mechanism can greatly increase the performance of a model in a wide range of different applications [49]. A GCN is limited by the fact that updates are always symmetrical over adjacent nodes. Introducing an Attention Mechanism into a GCN makes the network more complex and flexible, turning it into a so called Graph Attention Network (GATs). Previous work has shown [53] that for the smart background filter, a major advantage of this model is that it is extremely adaptive to differently shaped graphs. This makes the model good at dealing with unseen graph structures and generalizes well to new data.

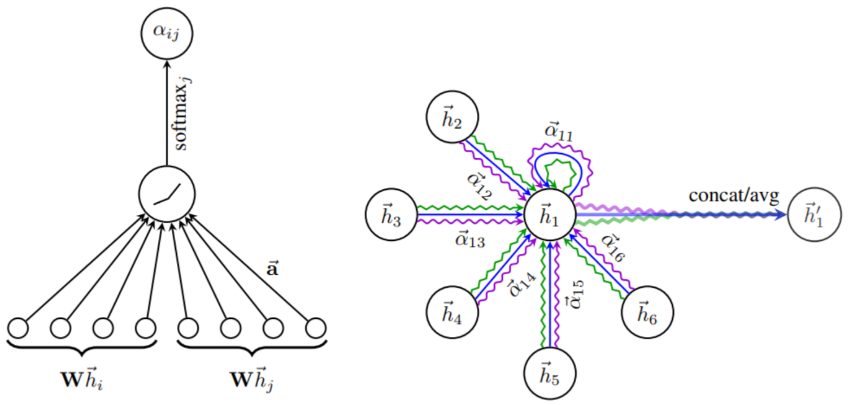


Figure 3.11: Left: The Attention Mechanism for GAT; Right: Multihead Attention

The attention mechanism is defined as the mapping $\vec{a} \in \mathbb{R}^{2F'}$ of two neighbouring nodes $\vec{h}_i, \vec{h}_j \in \mathbb{R}^F$ to a real number as show on the left in figure 4.5. The coefficients a_{ij} are defined by:

$$\alpha_{ij} = \text{softmax}_j \left(\sigma \left(\vec{a}^T \left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j \right] \right) \right) \tag{3.10}$$

with a global weight matrix $\mathbf{W} : \mathbb{R}^F \to \mathbb{R}^{F'}$, where F' is the dimension of the output feature and σ the activation function. The symbol $||$ represents the concatenation operation. The coefficients a_{ij} can be seen as a parameter measuring how much attention the network should pay to a given edge between node i and j. Combining all the coefficients, the update rule boils down to:

$$\vec{h}_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}^{(l)} \vec{h}_j^{(l)} \tag{3.11}$$

Another great property of the GAT model is that it allows multiple heads, adding additional complexity. In this context, multihead means that there are several GAT modules trained on the same graph. These GCNs are parallelized in different Attention Heads (figure 4.5 on the right). In addition, visualizing the Attention Mechanism can even help to understand the information propagation better.

Chapter 4

Training Data Production and Neural Network Architecture

To create a smart background filter, first and foremost two components are essential. On the one hand, the training data has to be created and on the other hand, the neural network architecture has to be defined. In this chapter, I will introduce the procedure of creating training data to fit to the model. In addition, I will define the exact model architecture used within the scope of this research.

4.1 Training Data Production

In this study, the target is to train the neural network filter with data from the decay chain $B^0 \to K^{*0}\nu\hat{\nu}$. The goal of this thesis is to study the effect analysis specific filters have on the performance of the background Monte Carlo filter and to develop an efficient framework so that the filter can efficiently be optimized for any desired study. For this, multiple different skims have been applied to the raw data in the skimming stage. At Belle II, skims are defined as a collection of high level analysis scripts, that truncate collected data to manageable sizes by applying certain selections. In other words, the output of the event generation, detector simulation and finally the reconstruction stages are the inputs to the skim. This raw data is saved into what is called mini data—summary tables (MDSTs). The output of a skim usually contains more information, but fewer events, by adding particle—candidate information, depending on what type of skim is utilized. In this study the FEI skim is used (see 1.6). The FEI skim adds additional variables, such as M_{bc} and can be used in many succeeding analyses. With the way the Belle II collaboration is setup, given the vast volume of MC data and that many analyses make use of the FEI, it is practical to commission a single global processing of background Monte Carlo with loose selections applied to produce the subset of output files containing reconstructed B_{tag} candidates. Generally speaking, the output of any skims are stored in so called user data–summary tables (UDSTs) as shown in figure 4.1. As a starting point for this study, the `feiHadronicB0` skim for the neutral (B^0) hadronic reconstructed channel has been analyzed. This skim applies a selection of event level cuts, to further decrease computational complexity [1]:

- $n_{\text{cleaned tracks}} \geq 3$
- $n_{\text{cleaned ECL clusters}} \geq 3$
- Visible energy of event (CMS frame) $> 4\text{GeV}$
- $2\text{GeV} < E_{\text{cleaned tracks and clusters in ECL}} < 7\text{GeV}$

where cleaned tracks and clusters are defined as:

- Cleaned tracks: $d_0 < 0.5\text{cm}, |z_0| < 2\text{cm}$ and $p_T > 0.1\text{GeV}$
- Cleaned ECL clusters: $0.296706° < \theta < 2.61799°$ and $E > 0.1\text{GeV}$

Once the FEI has been processed, the following cuts are applied on the B_{tag}:

- $M_{bc} > 5.24\text{GeV}$
- $|\Delta E| < 0.2\text{GeV}$
- SigProb > 0.001.

Here M_{bc} is the beam—constrained mass of the reconstructed B_{tag} candidate defined as $M_{bc} = \sqrt{E^2_{\text{beam}} + p^2_{B_{tag}}}$. The $|\Delta E|$ is the reconstructed energy difference $\Delta E = E_{B_{tag}} - E_{\text{beam}}$. Finally, the signal probability (SigProb) is an output of the FEI and indicates the correct reconstruction probability.

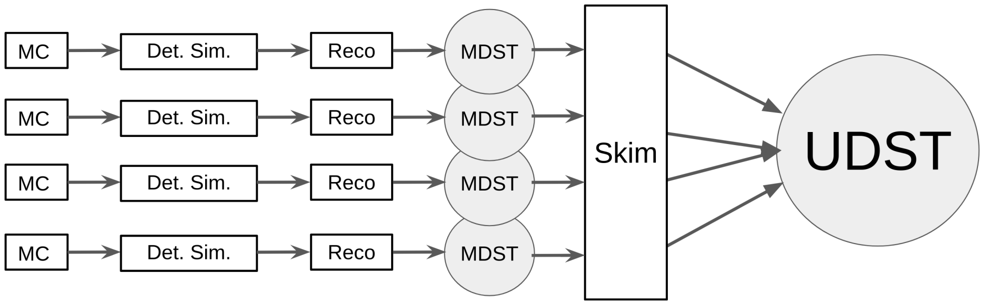


Figure 4.1: Schematic drawing of the skimming process, where one starts with the event generation and ends up with UDSTs.

4.1.1 Analysis Specific Skims

For this particular study, mixed and continuum background events are considered, which are part of the 14th MC production campaign by the Belle II collaboration, also referred to as the `MC14` dataset. For now it is only important to mention that already with this central loose skim around 94% of events are trivially thrown away. The mixed dataset is constructed from simulated $\Upsilon(4S) \to B^0\bar{B}^0$ events, while the continuum is made out of $e^+e^- \to q\bar{q}$ events.

Due to the vast amounts of data in the `MC14` dataset and the computational complexity of the FEI skim, I have decided to use preprocessed UDSTs as a starting point and add additional cuts on this loose but available selection. The production of the final labeled training data is done in three separate stages summarized in figure 4.2:

1. The `feiHadronicB0` skim is used to select events from MDSTs. This is done centrally within the Belle II collaboration. This process returns two different type of events, events that have failed this initial selection, which I will call *fail* events from this point on, and events that have survived this skim, so called *pass* events. These two types of events are then temporarily saved into different data–objects to be able to process them further. Pass events can thus easily be created by considering the already processed UDSTs and processing them.

2. The pass events from the last step need to be processed again to add analysis specific filters. For this, variables are calculated from the UDSTs, which are comprised out of the MC particle records. This is the data that in the end is used for training the NN. Furthermore, event level variables, which are variables like the event shape or event kinematics, are used for further cuts and validations and need to be computed too. Finally, B–variables from the FEI and variables that correspond to specific reconstruction (for details see chapter 6.1) are used. Some of these variables can subsequently be used to apply any additional filter, splitting up the pass events again into so called *UDST fail* and *pass* events. This distinction becomes important in the next step. To summarize, after two steps we have ended up with three different types of events, *Pass* events p that have survived both skims, *UDST fail* events f_u that have failed the analysis specific selection and passed the FEI skim and *MDST fail* events f_m that have failed the FEI.

3. In the third and last step, it has to be considered how exactly the failed data should be merged together. When training a machine learning model, it is desired to work with a balanced dataset. Thus we want to end up with data that is 50% comprised out of pass and 50% out of fail data. It is important to preserve the fraction $\frac{f_u}{f_m}$, as it is desired that the model can be generalized to new data and be used in production. f_u events usually preform differently to f_m events and the model should learn with the true distribution that occurs naturally in the Monte Carlo generation (more about this in section 5.5).

Finally, the pass and fail events get the label `True/False` attached to them respectively. All in all we now have ended up with labeled training data that can be directly used for training the NN.

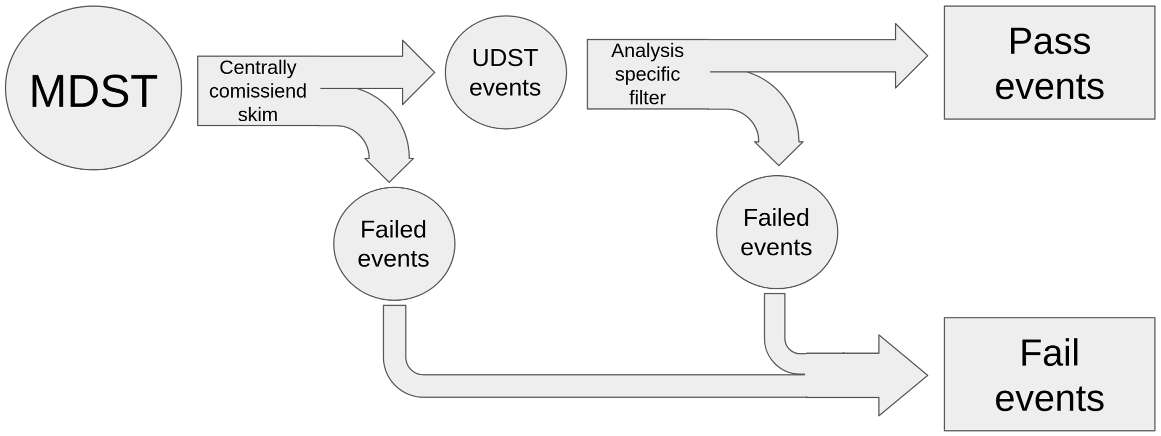


Figure 4.2: Complete overview of the training data production.

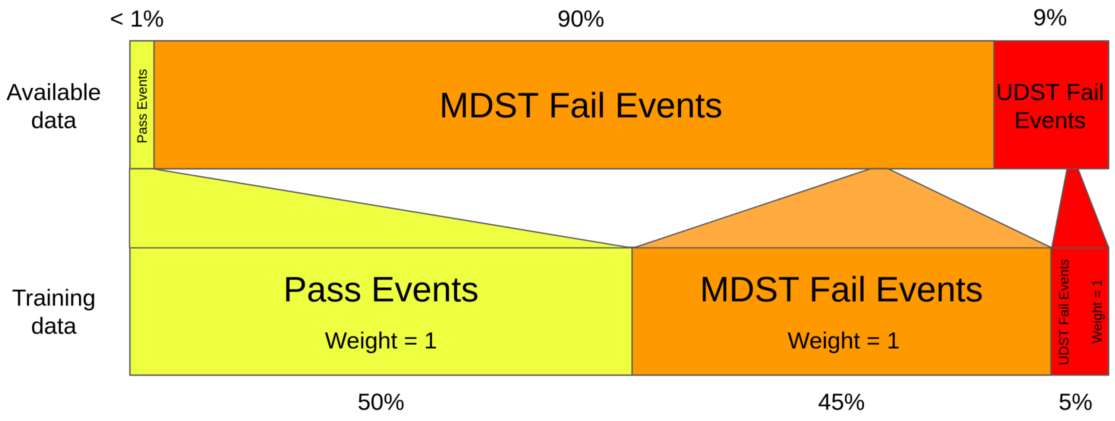


Figure 4.3: Usual distribution for unweighted training data. For a desired pass and fail distribution of 50%, most of the data in the fail region is discarded, which leads to strong undersampling.

4.2 Preprocessing

To prepare the data to be used with the NN filter, several features are chosen to be attached to each event. These variables then contribute to the training and evaluation process, while some other features are selected for a postprocessing analysis.

4.2.1 Feature Generation

Not all features that are present in the Monte Carlo dataset are an input to the network. It is sufficient to only select certain features to accurately represent a decay event. In general, the complete set of Monte Carlo generated information is similar to a rooted tree graph with particles as nodes and relationships as edges. Each node carries all the information associated with the corresponding particle. An example of a single $\Upsilon(4S)$ decay event is shown in figure 4.4.

```
 1   300553 (Upsilon(4S))     E: 1.100e+01 m: 1.058e+01 p:(...) v:(...)
 2       521 (B+)             E: 5.511e+00 m: 5.279e+00 p:(...) v:(...)
 4        -421 (anti-D0)      E: 1.939e+00 m: 1.865e+00 p:(...) v:(...)
10           310 (K_S0)       E: 5.913e-01 m: 4.976e-01 p:(...) v:(...)
20             211 (pi+)      E: 2.379e-01 m: 1.396e-01 p:(...) v:(...)
21            -211 (pi-)      E: 3.534e-01 m: 1.396e-01 p:(...) v:(...)
11           211 (pi+)        E: 3.484e-01 m: 1.396e-01 p:(...) v:(...)
12          -211 (pi-)        E: 7.756e-01 m: 1.396e-01 p:(...) v:(...)
13           111 (pi0)        E: 2.234e-01 m: 1.350e-01 p:(...) v:(...)
26              22 (gamma)    E: 1.660e-01 m: 0.000e+00 p:(...) v:(...)
27              22 (gamma)    E: 5.742e-02 m: 0.000e+00 p:(...) v:(...)
 5         413 (D*+)          E: 2.453e+00 m: 2.010e+00 p:(...) v:(...)
14           411 (D+)         E: 2.239e+00 m: 1.870e+00 p:(...) v:(...)
28            -321 (K-)       E: 8.801e-01 m: 4.937e-01 p:(...) v:(...)
29             211 (pi+)      E: 5.853e-01 m: 1.396e-01 p:(...) v:(...)
30             211 (pi+)      E: 7.737e-01 m: 1.396e-01 p:(...) v:(...)
15            22 (gamma)      E: 2.138e-01 m: 0.000e+00 p:(...) v:(...)
 6         313 (K*0)          E: 1.111e+00 m: 8.617e-01 p:(...) v:(...)
16           321 (K+)         E: 8.508e-01 m: 4.937e-01 p:(...) v:(...)
17          -211 (pi-)        E: 2.683e-01 m: 1.396e-01 p:(...) v:(...)
 3      -521 (B-)             E: 5.493e+00 m: 5.279e+00 p:(...) v:(...)
 7          43 (Xu0)          E: 2.052e+00 m: 8.646e-01 p:(...) v:(...)
18           211 (pi+)        E: 4.797e-01 m: 1.396e-01 p:(...) v:(...)
19          -211 (pi-)        E: 1.572e+00 m: 1.396e-01 p:(...) v:(...)
 8          11 (e-)           E: 1.621e+00 m: 5.110e-04 p:(...) v:(...)
 9         -12 (anti-nu_e)    E: 1.820e+00 m: 0.000e+00 p:(...) v:(...)
```

Figure 4.4: $\Upsilon(4S)$ decay structure within an event. The first two columns represent the array index and particle information (PDG code and human readable name) respectively. Here the indentation level corresponds to the depth of the particle in the decay tree. All particles in mixed events originate from the $\Upsilon(4S)$ The remaining columns shows data for each individual particle.

The Generated Variables are the building blocks of the input for the graph neural network structure, extracting the information to distinguish between background and signal events. These features are collected over all nodes from the graphs to calculate global features. The following are the Generated Variables:

- **PDG ID:** Officially referred to as the PDG particle numbering scheme by the Particle Data Group [51]. Each type of particle, including all elementary particles, atomic nuclei, composite particles and any particle beyond the Standard Model has a unique code. It ranges from 1 to ± 1000020040, with particles and antiparticles having the same absolute value but different sign. The range is too large to one–hot encode, therefore they are tokenized from 1 to the number of unique PDG IDs. These tokenized values are fed into the decay graph nodes representing the corresponding particles.
- **Mother array index:** The array index of the mother particle. This is used to build a mathematical representation of the graph.
- **Production Time:** Time that has passed from the initial $\Upsilon(4S)$ generation to the production of each particle.
- **Energy:** Energy of the particle in GeV.
- **Momentum:** Momentum p_x, p_y, p_z in GeV.
- **Production vertex position:** Euclidean coordinates of each particle in m. Any particle where one of the coordinates is larger than 10m are discarded due to the fact that they are created outside of the detector and will not provide any information.

4.2.2 Graphs and Batches

With the training data ready, it is time to build the graph neural network. With the help of the **mother array index**, the graphs are built up. This is either a graph with the root being $\Upsilon(4S)$ for the mixed data or a Z_0 for continuum data. Each node in the graph contains all information about itself. This includes all preprocessed generated variables, but also its physical observables like mass or charge. Each decay can be completely described by a single graph, which is matched to its corresponding label, whether or not this particular decay is going to pass the skim. After the construction of all the graphs, all events are randomly shuffled and and packed into batches with user defined sizes.

4.3 Neural Network Architecture

This chapter will explain the neural network architecture used in the analysis and its corresponding hyperparameters of the filter that was developed in the previous study [53]. Here it was displayed that the Graph Attention Network with Global Attention Pooling (GATGAP) architecture performs the best. The GATGAP architecture is a combination of a Graph Attention Network and Global Attention Pooling. After the preprocessing described in section 4.2, the PDG IDs are propagated through an embedding layer that maps them to a fixed dimensional tensor. The output of this is then concatenated with other generated features to form the complete dataset required for the NN. In addition,

			PDG	mass	charge	energy	prodTime	x
label	evtNum	arrayIndex						
False	33048021	0	300553	10.586205	0.0	11.013892	0.000000	-0.050857
		1	511	5.279650	0.0	5.501769	0.000000	-0.050857
		2	-511	5.279650	0.0	5.512124	0.000000	-0.050857
		3	411	1.869620	1.0	1.886227	0.000235	-0.050869
		4	321	0.493680	1.0	1.065091	0.000235	-0.050869
		5	-211	0.139570	-1.0	0.230585	0.000235	-0.050869
		6	-423	2.006980	0.0	2.076368	0.000235	-0.050869
		7	-211	0.139570	-1.0	0.243499	0.000235	-0.050869
		8	413	2.010280	1.0	2.625607	0.000131	-0.050525
		9	-211	0.139570	-1.0	0.364729	0.000131	-0.050525
		10	221	0.547850	0.0	1.344311	0.000131	-0.050525
		11	111	0.134980	0.0	1.177473	0.000131	-0.050525
		12	310	0.497611	0.0	0.980574	0.004980	-0.063048
		13	211	0.139570	1.0	0.295891	0.004980	-0.063048
		14	111	0.134977	0.0	0.609761	0.004980	-0.063048

Table 4.1: Example snippet of the Pandas dataframe output of the feature generation. The first ten particles of a single fail event are shown, sorted by array index with the MCParticles features truncated for demonstration purposes.

the graph represented by a list of edges provides all the information about how the nodes are connected. Finally, the graph including the node features represent complete events and can be passed through the graph neural network layers. These layers then learn the structure of the graphs with its nodes and are repeated multiple times. The number of modules is controlled by the hyperparameter *number of layers*.

The initial node features are input to the first layer. After this, each layer outputs a new set of features for each node. This means that the GATModules on the one hand use the output of the previous layer as inputs and on the other hand will send their outputs into the pooling layer. This will collect the features from the complete set of nodes in the graph and form global features that are not directly dependent on the graph structure. In other words, for any graph shape the global features stay the same. Global Attention Pooling adds a layer of complexity to the pooling process. Compared to a regular pooling layer, Global Attention Pooling performs a weighted sum over nodes, similarly to Graph Attention where weights are attached to the edges of the graph. Finally, after propagating through all layers, a fully connected dense layer summarizes the information into a one dimensional score $p \in \mathbb{R}^1$. The complete structure is of the model is shown in figure 4.5. Here a architecture with 3 layers is presented.

The tunable hyperparameters of the model are:

- *The number of layers*: The number of GAT Modules in the neural network.
- *The number of heads*: The number of attention heads in each GATModule.
- *The number of units*: The size of each hidden state in the Dense and GAT layers. Within each model, it defines the size of global features.

In section 6.2, I describe a streamlined process for optimizing these hyperparameters for any analysis specific skim.

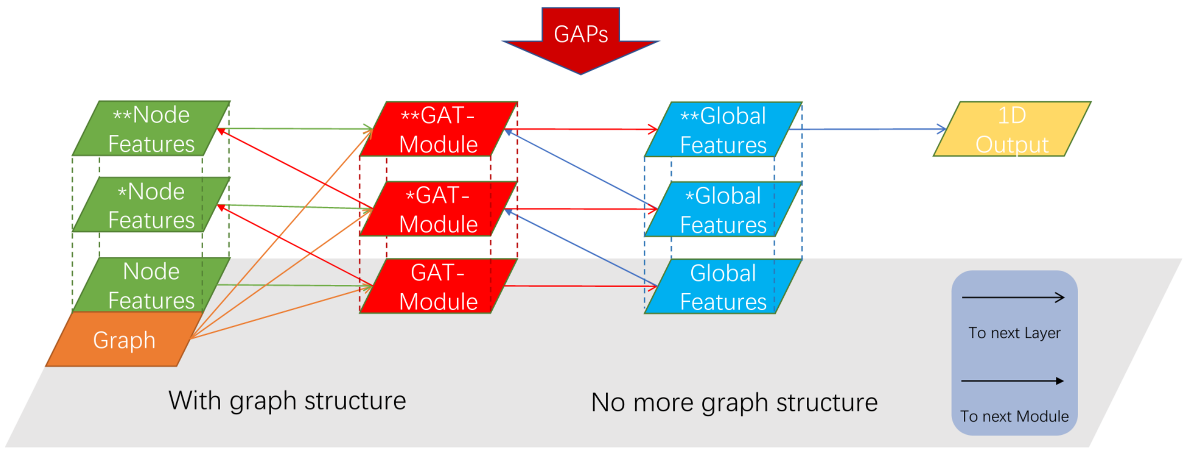


Figure 4.5: Network architecture of the GATGAP model used in the scope of this study. Other architectures have also been used in past studies, but it has been shown that this architecture yields the best performance [53].

Chapter 5

Performance Measure and Training

Now that we have the general architecture and the training data of our model, the next step is to consider how exactly the network should be trained. Which performance measure p should be used? In this chapter the idea behind the speedup loss is introduced. This is then extended to use cost sensitive learning, where weighted training data for extremely low retention rates is used.

The output of the NN filter is a real number $p \in [0,1] \subset \mathbb{R}$. This value indicates the probability predicted by the filter that the corresponding event is going to pass a specific skim. The higher the value, the more likely the filter thinks an event is going to pass. As I will show, maximizing the accuracy of the model and cutting the output by a threshold yields unusable biased results. Instead, by sampling the NN output in a certain way, the bias can be mitigated and the model can be trained to better improve the simulation speed by adjusting the performance metric. Finally, I will show that due to the nature of the distribution of the training data, namely that the retention rates get extremely low and the number of pass events is limited, using a balanced distribution of pass and fail data returns suboptimal performance. Since most ML algorithms assume the underlying data is balanced, a mix between undersampling of the majority class (fail events) and cost–sensitive learning is used to balance the data.

5.1 Binary Cross–Entropy

Usually when the task is optimizing the accuracy on a binary classification problem, the binary cross–entropy measure is used [18]. The cross–entropy between two probability distributions based on the same underlying set of events is a measure how different the distributions are. It returns the average information needed in the classification of an event from the test set. The more information that is needed of an event, the more dissimilar the distributions are.

The cross entropy for two discrete probability distributions p and q with the same support χ is defined as:

		True labels		
		Pass	Fail	Total
Predicted labels	Pass	N_{TP}	N_{FP}	$N_{FP} + N_{FP}$
	Fail	N_{FN}	N_{TN}	$N_{FN} + N_{TN}$
	Total	$N_{TP} + N_{FN}$	$N_{FP} + N_{TN}$	N

Table 5.1: Evaluation of the NN on the training data. Shown is a visual representation of all 4 different cases with neural network output for classification. Green highlights the cases where the classification has been successful, red indicates false negatives which can introduce a bias and gray represents false positives. While false positives do not introduce a bias, in the case of the smart background simulations, the speedup of the simulation will be reduced as some events are simulated unnecessarily.

$$\mathcal{H}_\chi(p,q) = -\sum_{i\in\chi} p(i)\log(q(i)) \tag{5.1}$$

For the binary case, this function can be simplified. Here the binary cross entropy can be expressed at:

$$\mathcal{H}(y_\text{true}, y_\text{pred}) = -\frac{1}{N}\sum_{i=1}^{N}(y_\text{true}(i)\log(y_\text{pred}(i)) + (1-y_\text{true}(i))\log(1-y_\text{pred}(i))), \tag{5.2}$$

where N is the number of events in the dataset, y_true the set of true labels and y_pred the predicted probabilities for classification (the NN output) for all events in the dataset.

5.2 ROC curve

After a NN is trained with the (binary) cross–entropy as loss, the network is optimized to classify events with the highest possible accuracy. There are 4 potential outcomes of the NN when classifying an event:

- **True positive (TP):** The event is classified as true and is actually true.
- **False positive (FP):** The event is classified as true and is actually false.
- **False negative (FN):** The event is classified as false and is actually true.
- **True negative (TN):** The event is classified as false and is actually false.

A visual overview of these cases is shown in table 5.1. To get a better understanding of the best threshold for the NN, we can use the receiver operating characteristic (ROC) curve to visualize the results. It is crucial for evaluating the quality of binary classifications. The

ROC curve is generated by plotting the rate of the true positives against the rate of false positives:

$$\begin{aligned} \text{True positive rate:} \quad TPR &= \frac{TP}{TP+FN} = \frac{TP}{P} \\ \text{False positive rate:} \quad FPR &= \frac{FP}{FP+TN} = \frac{FP}{N} \end{aligned} \tag{5.3}$$

The area-under-curve (AUC) score provides a quantitative measure to compare the performance of different networks. A random classifier which is unable to learn from the dataset and can not classify between the two classes is represented by a straight line from the bottom let to the top right as shown in figure 5.1. Any classifier should be located above this line, the further the better. A perfect classifier which is able to predict all labels correctly, from a triangle with the top right corner at the top left corner. The AUC score quantifies this performance. In non continuous discrete cases it sums up the area under the curve and ranges from 0.5 (random guessing) to 1 (perfect classifier). For classification, the aim is to maximize this measure.

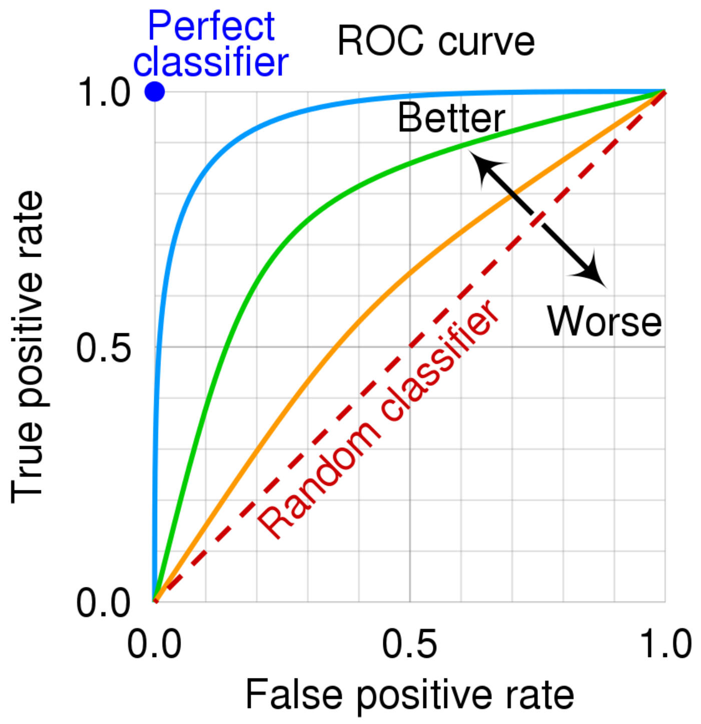


Figure 5.1: The ROC curve comparing a more to a less accurate model.

5.3 Evaluation

5.3.1 Bias and the Effect of Different Cuts.

Training the model defined in chapter 4.3 with two different training sets, comprised out of the original FEI skim and a generic tighter analysis specific skim, significant performance differences can already be noticed. As can be seen in figure 5.7, the AUC for the generic tighter skim is much higher than the AUC for the FEI skim, using the same model and amount of training data. However, figure 6.4 shows a major flaw of the smart background simulations. Namely, a significant bias is introduced in the final data by the filter. In the context of HEP simulations, physicist are often more interested in the distribution of specific observables, rather than detailed information about individual events and particles. This is obviously a problem, as any bias will skew any subsequent physics analysis, making the simulated data unreliable and in many cases basically unusable. The intrinsic reason behind this situation are the false negatives the filter produces. It is evident that any correctly identified event does not alter the final distributions. In addition, false positives are also not problematic, as these events will be taken care of at the end of Monte Carlo simulation in the skimming stage. However, false negatives are troublesome. The filter will discard events which we can not get back at a later stage. In essence, the events that are falsely classified at failing the skim will end up missing in any physics analysis.

The theses by Bross [13] and Yu [53] have addressed this problem in various ways. The most effective approach for mitigating the bias is a procedure based on importance sampling, which I will introduce in the next section.

5.3.2 Importance Sampling

In general, sampling is a method of selecting a subset of a large dataset in a certain manner to increase the statical meaning of any analysis of the data. In practice, any distribution can be biased if the samples are not selected at random and therefore will decrease the reliability of any subsequent analysis [8]. As I have shown in section 5.3.1, the neural network filter output introduces a bias in many observables. With the help of sampling, it is possible to adjust the distributions of all observables in such a way, that the bias is mitigated. This is even possible without the need that every single event is correctly classified.

There are many different techniques for sampling:

- Random sampling: The most basic and simple method of sampling. Like the name suggests, events are selected at random.
- Metropolis Hastings: Based on Markov chain Monte Carlo (MCMC) methods. It is used for obtaining sequences of random samples from probability distributions for which direct sampling is not possible.

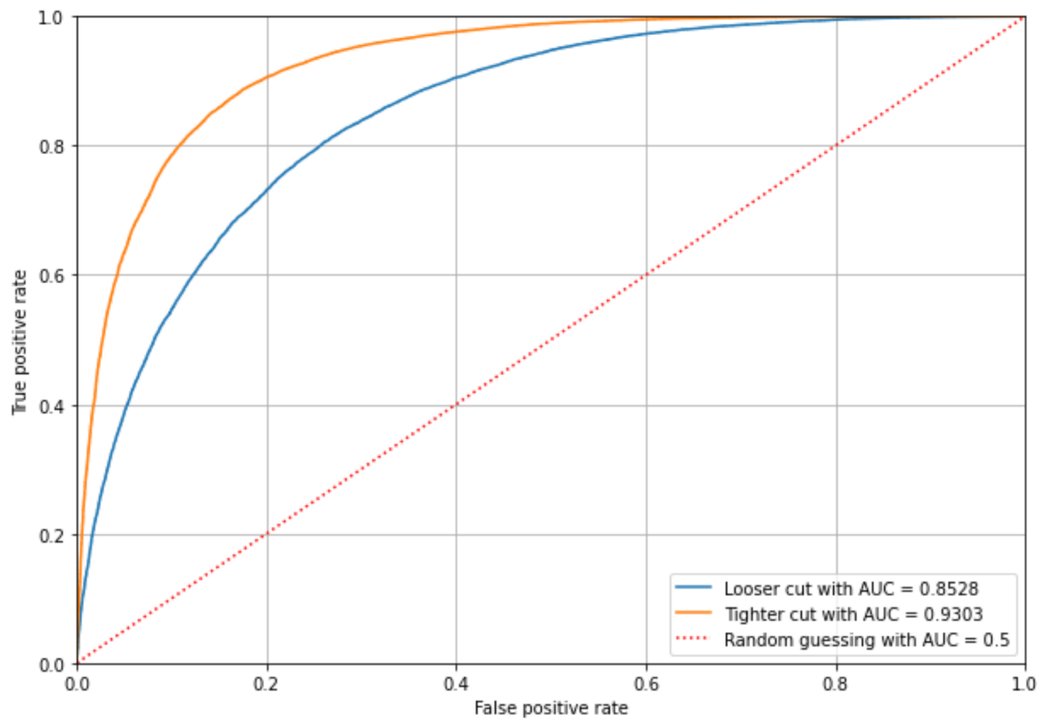


Figure 5.2: Comparison of the AUC value between a generic tighter and looser cut.

- Bootstrap sampling: Bootstrapping is a sort of random sampling with replacement, and falls under the broader class of resampling methods. Random forests rely on this method.

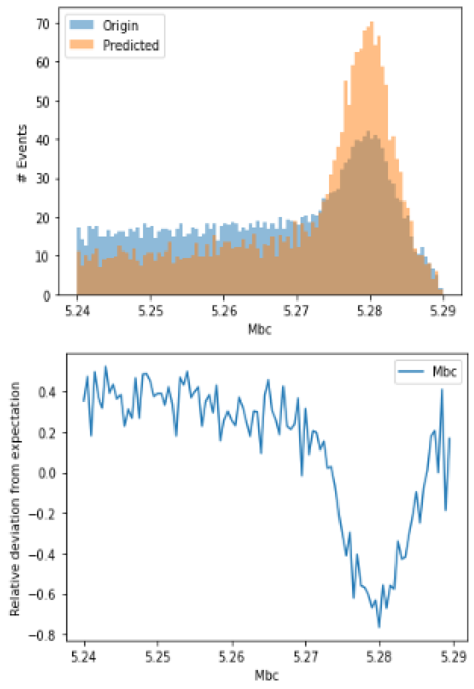
- Importance sampling: The method of sampling that is used in the neural network filter.

Like the name suggests, importance sampling procedure is a method to sample events from a dataset based on their importance. In this section I will closely

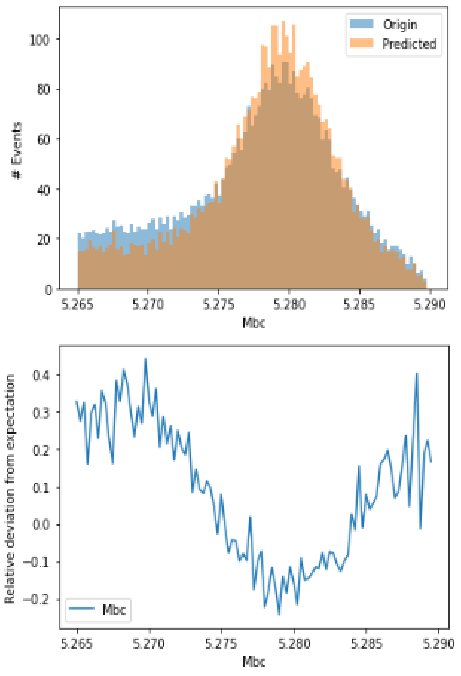
follow the reasoning from [40]. The importance of an event is defined as the probability of the event being selected. We have the following expression of the expectation value $E(F(x))$:

$$E(f(x)) = \int_{\chi} f(x)p(x)dx, \tag{5.4}$$

where χ is the support of the random variable X, $f(x)$ is the function to be sampled and $p(x)$ is the PDF. In regular random sampling, unlikely events in the χ space are selected equally likely, independent of their PDF and thus contribute only little to the final result $E(F(x))$. In practice, this wastes a lot of computational resources. Instead, focusing on regions in χ that are more important might reduce the time complexity. In Monte Carlo computation with high dimensional models, samples are generated with the following Importance Sampling algorithm:



(a) Neural network filter output applied on the regular FEI skim described in chapter 4.1.



(b) Neural network filter output applied a tighter analysis skim. In this specific example, additional cuts have been applied to M_{bc} and the sigProb.

Figure 5.3: Effect of a generic example cut on the output of the neural network filter. In the first image the M_{bc} distribution after simulation with the filter is shown, comparing the filtered data to the original. The second row shows the relative deviations from the data that has been simulated using the smart background filter and the original dataset. For both the looser and tighter cut, the same model hyperparameters and training data size have been used.

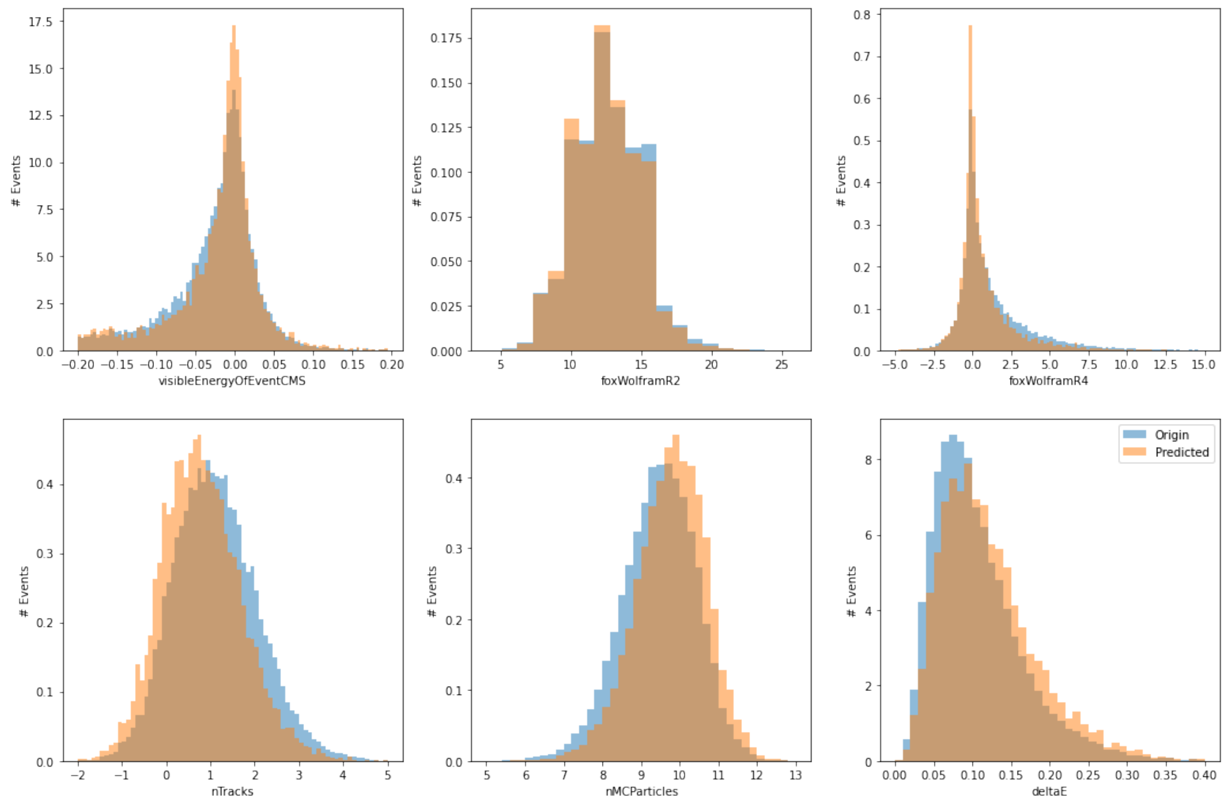


Figure 5.4: Examples of biases present in the observalbes of the filtered data.

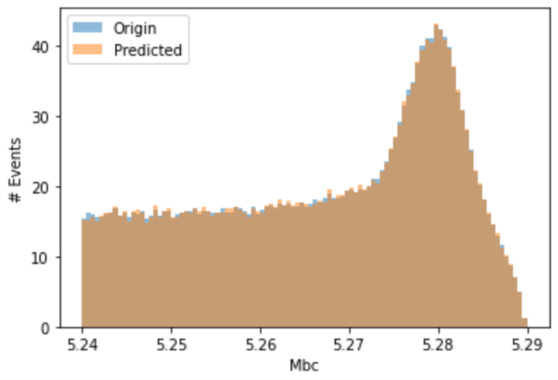
1. Draw a sample $\vec{x}$ from a trial distribution $g(\vec{x})$
2. Calculate the importance of the events in the sample

$$\omega^j = \frac{p(x^j)}{g(x^j)}, \quad \text{for all} \quad j = 1, \ldots, \dim(\vec{x})$$

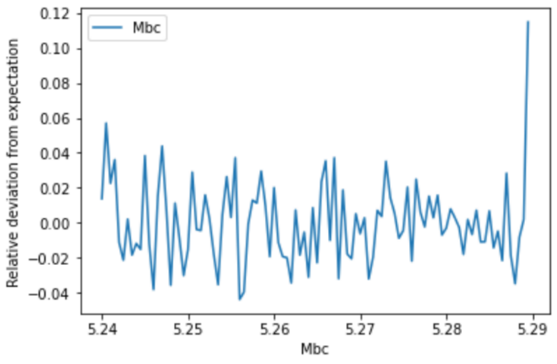
3. Approximate the desired expectation value $E(f(X))$ by the weighted average of the samples:

$$\hat{E}(f(x)) = \frac{\sum_{j=1}^{N} \omega^j f(x^j)}{\sum_{j=1}^{N} \omega^j}$$

To reduce the estimation error $\hat{E} - E$ as much as possible, the sample distribution $g(x)$ should be chosen so that it is as similar as possible to the original shape $f(x)p(x)$. For the smart background filter, the sample distribution is constructed by the NN output. A certain event is selected with a probability of the value of the NN output $p_{NN}(x)$. When considering every event passing the skim $N = \sum_x p_{\text{skim}}(x)p_i(x)$, where $p_i(x)$ represents the distribution of all observables in the events that the event generation process attempts to simulate. Due to the fact that the event detector simulation and reconstruction stages are



(a) Histogram of the original and simulated distribution of M_{bc} using importance sampling.



(b) Deviations between the original and simulated distribution.

Figure 5.5: Bias mitigation using importance sampling. Deviations for all physical observables are minimal.

computationally extremely expensive, evaluation $p_{\text{skim}}(x)$ is costly. Therefore instead of $p_{\text{skim}}(x)$, the NN output is $p_{NN}(x)$ is added in the sampling. For this an importance weight is attached to each events:

$$\omega(x) = \frac{p_i(x)}{p_i(x)p_{NN}(x)} = \frac{1}{p_{NN}(x)} \tag{5.5}$$

Intuitively, where the NN filter assigns a low probability for passing the skim will be sampled less frequently. However, if such an event does pass the filter, the importance weight will be high and this particular event will influence the final distribution more.

Figure 5.5a shows the original and simulated distribution of M_{bc} using importance sampling. The deviations between the original and simulated distribution are shown in Figure 5.5b. In fact, by construction the bias is completely eliminated for all physical observables.

5.4 The Speedup Metric

Taking a step back and again considering what exactly the task T of the ML model should be, it becomes clear that maximizing the accuracy of classifying events is not necessary the primary objective. Instead, the purpose of the smart background simulations is to increase the speed of the simulation as much as possible. Since it is necessary to work with weighted events to mitigate the bias, a one to one comparison of a original and simulated dataset is not possible. For weighted events, the metric *effective sample size* N_{eff} is defined as the number of events that would yield the same amount of statistical uncertainty as the case for the original unweighted dataset. Thus, the speedup is generally defined as the ratio between the time consumption of the complete work flow with and without the NN filter to produce the same amount of statistics or effective sample size.

5.4.1 Speedup for the Sampling Method

The goal of this section is to derive an expression for the Speedup metric, which is defined as the ratio of time it takes to simulate the same statistics with and without the NN filter. The sampling method is a method that combines the speedup metric with importance sampling. By using importance sampling, uncertainty is introduced by the random selection of events, where events take their NN output p as probabilities to be kept. For each event that is selected, an uncertainty of $\frac{1}{p}$ is added, which is equal to the weight for the sampling method:

$$\omega(x) = \frac{1}{p(x)}. \tag{5.6}$$

With this information, the total statistical uncertainty for weighted events can be expressed in the form of:

$$S_{\text{total}} = \sqrt{\sum_{j=1|t_j=\text{Pass}}^{N_{\text{Pass}}} \omega_j^2 p_j} = \sqrt{\sum_{j=1|t_j=\text{Pass}}^{N_{Pass}} \frac{1}{p_j}}. \tag{5.7}$$

where p_j is the NN output and t the label for the event j. The expression $t_j = \text{Pass}$ in the sum means that only events that have passed the filter are selected. The effective sample size is proportional to the squared relative statistical uncertainty

$$N_{\text{eff}} = \left(\frac{\sum_{j|t_j=\text{Pass}} \omega_j p_j}{S_{\text{total}}}\right)^2 = \frac{(\sum_{j|t_j=\text{Pass}} \omega_i p_i)^2}{\sum_{j|t_j=\text{Pass}} \omega_i^2 p_i} = \frac{N_{\text{Pass}}^2}{\sum_{j|t_j=\text{Pass}} w_j}, \tag{5.8}$$

with the total number of pass events N_{Pass}.
However, the sample size can also be expressed by the NN scores directly. The number of events that can pass the NN filter is trivially given by:

$$N_{\text{filter}} = \sum_j p_j, \tag{5.9}$$

for large enough number of events. The Speedup metric is designed to be evaluated on a sample consisting of a balanced dataset, meaning that there is an equal amount of pass and fail data. Because of this, the retention rate r has be considered. Equations 5.24 and 5.9 must be adjusted to:

$$\begin{aligned} N'_{\text{eff}} &= rN_{\text{eff}} \\ N_{\text{filter}} &= rN_{\text{TP}} + (1-r)N_{\text{FP}} \\ &= r\sum_{\{i|t_i=\text{ Pass }\}} p_i + (1-r)\sum_{\{i|t_i=\text{ Fail }\}} p_i, \end{aligned} \tag{5.10}$$

to consider the amount of pass and fail data present in the real world. In addition, the effective sample size has to be altered to compensate for the retention rate. This can be

Stage	ms/event
Event generation t_{gen}	0.08
Neural network inference t_{NN}	0.63
Detector simulation and reconstruction t_{SR}	97.04

Table 5.2: Rough estimation of the time consumption of the complete work flow. The times are subject to change for different analyses and model archtecture.

done by increasing the effective sample size without the NN filter by a factor of $\frac{1}{r}$. The final effective sample size, can thus be quantified as:

$$N_{\text{No filter}} = \frac{1}{r} N'_{\text{eff}} = N_{\text{eff}}. \tag{5.11}$$

Combining equation 5.11 and 5.9, the effective speedup boils down to the ratio

$$\text{Speedup}_{\text{eff}} = \frac{N_{\text{No filter}}}{N_{\text{filter}}}. \tag{5.12}$$

This is a measure for the additional effort that has to be put in without the NN filter. However, this metric does not take the time complexity of the NN inference and event generation into account which is needed to calculate the actual final speedup of the simulation. A rough estimation of time consumption is given in table 5.2. Especially for larger models with more trainable parameters, the NN inference time is subject to change. However, these changes are small and mostly negligible. Moreover, previous studies have shown that even large changes in the values provided in table 5.2 have little to no effect on the speedup measure, meaning that the speedup metric is robust in variations in these parameters [53].

It has to be differentiated between the different types of predictions from the NN to accurately compute the time complexity of simulation with the filter. **True positive** and **false positive** predictions are the most time consuming. These events will be sent to the skimming and thus run through the expensive simulation and reconstruction phases:

$$T_{TP} = T_{FP} = t_{gen} + t_{NN} + t_{SR}. \tag{5.13}$$

On the other hand, **true negative** and **false negative** predictions are fast to simulate:

$$T_{TN} = T_{FN} = t_{gen} + t_{NN}. \tag{5.14}$$

The proportion between pass and fail events is r and $1-r$ respectively, while the number of pass and fail predictions from the NN can be expressed through their probabilities with p and $1-p$. Gathering all the information, the final expression for time consumption with the NN filter is given by:

$$\begin{aligned}
t_{\text{filter}} &= (N_{\text{TP}} + N_{\text{FP}}) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) + \\
&\quad (N_{\text{FN}} + N_{\text{TN}}) \cdot (t_{\text{gen}} + t_{\text{NN}}) \\
&= \left(r \sum_{\{i|t_i = \text{ Pass }\}} p_i + (1-r) \sum_{\{i|t_i = \text{ Fail }\}} p_i \right) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) + \\
&\quad \left(r \sum_{\{i|t_i = \text{ Pass }\}} (1-p_i) + (1-r) \sum_{\{i|t_i = \text{ Fail }\}} (1-p_i) \right) \cdot (t_{\text{gen}} + t_{\text{NN}})
\end{aligned} \tag{5.15}$$

Finally the time consumption of the Monte Carlo simulation without any filter is given by:

$$t_{\text{No filter}} = N_{\text{No filter}} \cdot (t_{\text{gen}} + t_{\text{NN}}) \,, \tag{5.16}$$

to get a final term for the Speedup:

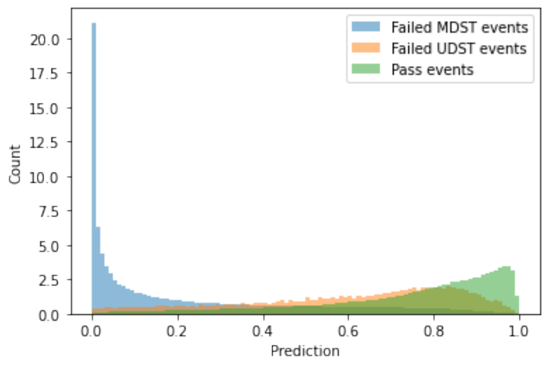
$$\text{Speedup} = \frac{t_{\text{No filter}}}{t_{\text{filter}}}. \tag{5.17}$$

5.5 Weighted Training Data

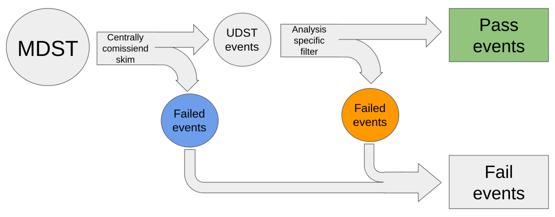
With the ML task thoroughly defined, it is important to analyze how the predictions of the NN behave with the speedup as loss function. Figure 5.6a shows the distribution of the NN output, separated by the event category. As shown in figure 5.6b, the green distribution represents events that have passed the skim, the orange distribution shows events that have passed the initial FEI skim but failed the analysis specific filter (UDST fail events) and finally the blue events show events that have failed the FEI skim (MDST fail events). It can clearly be seen that the UDST false events are much harder to distinguish from pass events than MDST false events. There are two major reasons for this:

1. The need alone to add a secondary selection for the UDST false events is a sign that these events look much closer to pass events than events that have been filtered with the FEI skim. The tighter the analysis specific cut is, the more similar the events look to pass events. This is why the NN filter has a harder time distinguishing pass events from these false events.

2. The second reason originates from the fact that the natural fraction of UDST false events over MDST false events has to be preserved in the training data so that the model can be generalized well to new unseen data (see section 4.1). Especially for hard analysis specific cuts with vanishing retention rates, the main limiting factor for training data is the availability of pass events. For conserving the right fractions, a large fraction of available UDST false events has to be discarded, the more the lower the retention rate. What makes this even worse is that, depending on the cut, no more than 10% of the training data are UDST false events as shown in figure 4.3. A concrete example of how a balanced dataset is calculated is given in table 5.4.



(a) NN output after training with the speedup metric separated by their type.



(b) Illustration of the origin of the categories in figure 5.6a.

Figure 5.6: Illustration, that with unweighted training data, not enough UDST false events are available to efficiently train the NN.

Both these factors combined lead to the poor performance of the NN filter on UDST false events. While the first reason is important, the fact that UDST false events and pass events are similar is a fact and can not be changed. However, it is possible to adjust the training of the NN to be able to better distinguish these events. The easiest way to do this is to increase the number of UDST false events the NN sees during training. This can be done by loading an unbalanced dataset, with a much larger proportion of UDST false events. However, to balance the data again, certain balancing weights must be attached to each event, to balance the whole set of training data to the original distributions. This is often referred to as cost–sensitive learning.

Typical cases of unbalanced data includes only two different types of events (e.g. fraud detection), where calculating the balancing weights is straightforward. Instead, to calculate the balancing weights for three different types of events, the balanced rates of the unweighted training data has to be derived. For this, two variables must be known:

- The total number UDST events U in the dataset. This is user defined and thus known beforehand.
- The FEI skim retention rate. This is the fraction of the number of UDST events U over the number of MDST minus UDST events M. This is defined for each FEI skim separately and is around $r = 10\%$ for the `feiHadronicB0` skim.

Using this information, we can calculate the number of total MDST events M:

$$\begin{aligned} \frac{M}{U} &= \frac{1-r}{r} \\ M &= \frac{U(1-r)}{r} \end{aligned} \tag{5.18}$$

Let X be the total amount of fail events needed to balance the dataset with 50% pass and fail events, then

$$X = U_f + M = k \cdot U + M, \tag{5.19}$$

where U_f is the number of UDST false events needed in the dataset and k the needed fraction of UDST false events. Combining equation 5.18 and 5.19 we get a final term for U_f:

$$\begin{aligned} U_f = X - M = X - \frac{U_f(1-r)}{k \cdot r} \\ = \frac{X}{(1 + (1/k) \cdot (1-r)/r)} \\ = \frac{X}{1 + (1-r)/(k \cdot r)} \end{aligned} \tag{5.20}$$

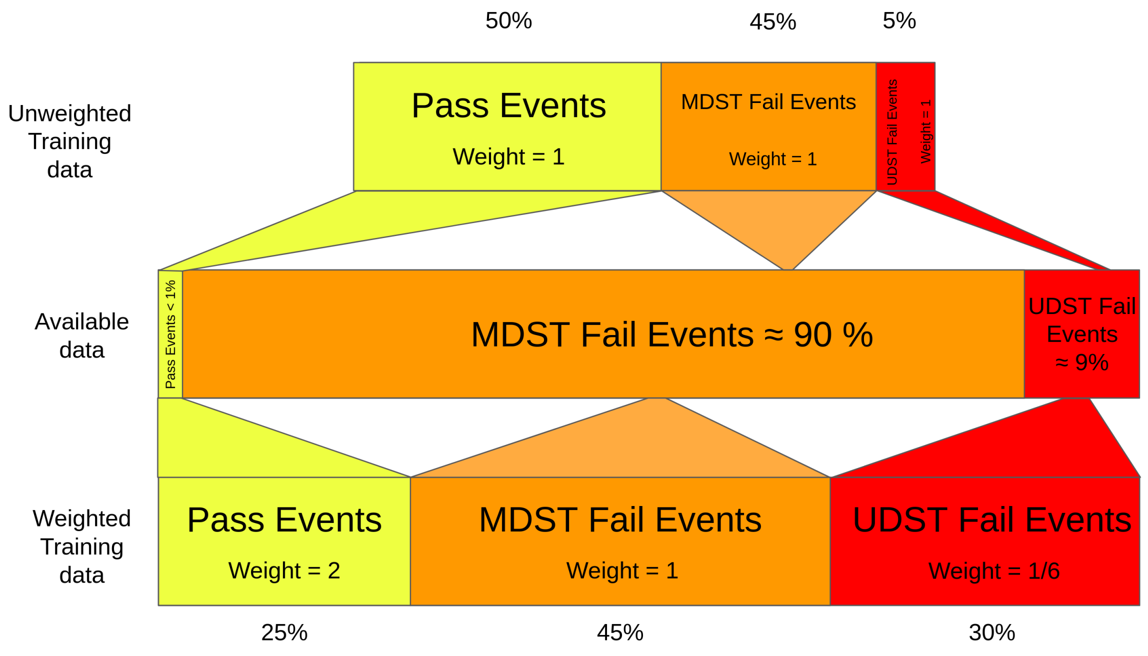


Figure 5.7: Schematic drawing of a comparison of the training data with and without balancing weights.

By knowing the exact number of each type of event that is needed to balance the dataset so that the distributions represent the natural distributions that would occur during simulation, weights can be calculated to scale any non–zero mix of UDST false events, MDST false events and pass events. Let N_{f_M}, N_{f_U} and N_p be the number of MDST false events, UDST false events and pass events present and let N_f be the total number of false

Variable	Description
r	Skim retention rate
k	The fraction of UDST fail events within the fail data for a balanced dataset
M	Number of MDST events
U	Number of UDST events
U_f	Number of UDST fail events
X	Total number of fail events
$\mathbf{w_{f_M}}$	Weights attached to MDST events
$\mathbf{w_{f_U}}$	Weights attached to UDST fail events
$\mathbf{w_p}$	Pass weights

Table 5.3: Summary of the variables used in the calculation of the weights.

events in the unbalanced dataset. Then the weights are:

$$\begin{aligned}\text{MDST weights:}\quad \mathbf{w_{f_M}} &= \frac{M}{N_{f_M}}\\ \text{UDST fail weights:}\quad \mathbf{w_{f_U}} &= \frac{U_f}{N_{f_U}}\\ \text{Pass weights:}\quad \mathbf{w_p} &= 1\end{aligned} \tag{5.21}$$

Due to the fact that we are not working with a balanced dataset anymore, the Speedup metric from section 5.4 is not applicable anymore and has to be adjusted. More specifically, to calculate the updated time complexity for the case with the NN filter the number of events in equation 5.15 for each category has to be altered:

$$\begin{aligned}N'_{\text{TP}} &= N_{\text{TP}} \cdot \mathbf{w_p} = r \sum_{\{i|t_i=\text{ Pass }\}} p_i \cdot \mathbf{w_p}\\ N'_{\text{FP}} &= N_{\text{FP}} \cdot \mathbf{w_{f_x}} = (1-r) \sum_{\{i|t_i=\text{ Fail }\}} p_i \cdot \mathbf{w_{f_x}}\\ N'_{\text{FN}} &= N_{\text{FN}} \cdot \mathbf{w_p} = r \sum_{\{i|t_i=\text{ Fail }\}} (1-p_i) \cdot \mathbf{w_p}\\ N'_{\text{TN}} &= N_{\text{TN}} \cdot \mathbf{w_{f_x}} = (1-r) \sum_{\{i|t_i=\text{ Fail }\}} (1-p_i) \cdot \mathbf{w_{f_x}},\end{aligned} \tag{5.22}$$

where $\mathbf{w_{f_x}} = w_{f_M} \cdot (1-k) + w_{f_U} \cdot k$, so that:

$$\begin{aligned}t'_{\text{filter}} =& (N'_{\text{TP}} + N'_{\text{FP}}) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) +\\ & (N'_{\text{FN}} + N'_{\text{TN}}) \cdot (t_{\text{gen}} + t_{\text{NN}})\end{aligned} \tag{5.23}$$

In addition, to calibrate the derivation of the time consumption without the NN filter, the effective sample size in equation 5.24 has to be adjusted. With the convention in this thesis $w_p = 1$, the simulation time for weighted and unweighted data in the case of no filter

Type	Variable	Quantity
User defined	Skim retention rate	$r = 0.1$
	number of Pass events	$N_p = 100$
	Analysis specific skim retention rate	$r_a = 0.01$
Calculated	total number of UDST events	$U = \frac{N_p}{r_a} r = 1,000$
	total number of MDST events	$M = \frac{U(1-r)}{r} = 9,000$
	UDST false fraction	$k = \frac{U_f}{M+U_f} = \frac{r-r_a}{1+r-r_a} = 0.08$
Balanced data	total number of fail events	$N_f = N_p = 100$
	number of UDST fail events	$N_{f_U} = N_f \cdot k = 8$
	number of MDST fail events	$N_{f_M} = N_f \cdot (1-k) = 92$

Table 5.4: Specific example of calculating a balanced dataset.

does not change. However, in the general case where $w_p \neq 1$, the effective sample size has to be changed in the following way:

$$N'_{\text{No filter}} = \frac{\left(\sum_{j|t_j=\text{Pass}} \mathbf{w_p}\right)^2}{\sum_{j|t_j=\text{Pass}} \frac{\mathbf{w_p}}{p_i}} = \frac{(N_{\text{Pass}} \cdot \mathbf{w_p})^2}{\sum_{j|t_j=\text{Pass}} \frac{\mathbf{w_p}}{p_i}}, \tag{5.24}$$

and thus:

$$t'_{\text{No filter}} = N'_{\text{No filter}} \cdot (t_{\text{gen}} + t_{\text{NN}}) \tag{5.25}$$

Finally, the general speedup for weighted training data is given by:

$$\text{Speedup} = \frac{t'_{\text{No filter}}}{t'_{\text{filter}}}. \tag{5.26}$$

Figure 5.8 shows how the speedup loss compares to the true speedup the NN would provide. Usually in ML problems, one defines losses as metrics to be minimized. It is evident that the speedup loss is the inverse true speedup which is the metric that we are most interested in. This means that from the loss, the performance in the real world can directly be interpreted.

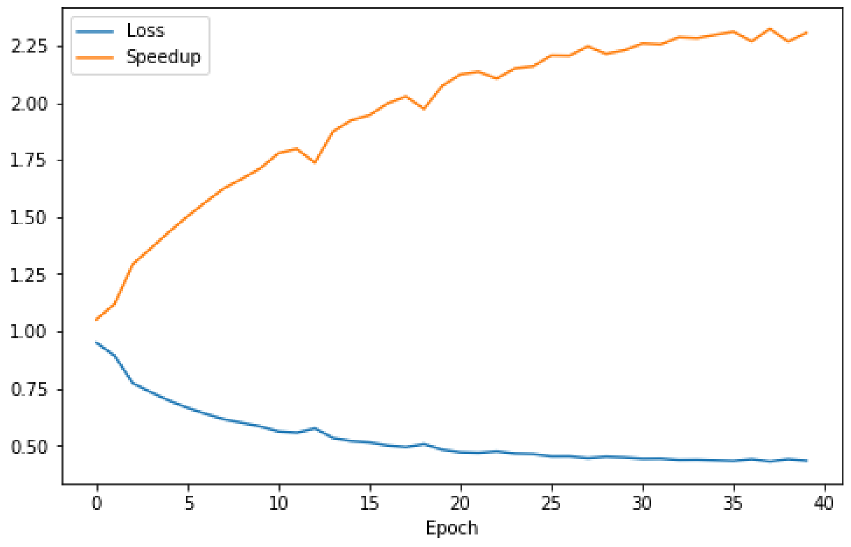


Figure 5.8: Comparison of the Speedup loss and the true speedup of the NN filter

Chapter 6

Experimental Procedure and Results

In a potential physics analysis of the $B^0 \to K^{*0}\nu\bar{\nu}$ process, the signal must be identified amongst all e^+e^- collision events. To achieve this, the Monte Carlo simulations are used to tune the analysis for identifying and isolating the signal from the background. Especially for this decay, with two invisible neutrinos in the final state, information about both B mesons in the $e^+e^- \to \Upsilon(4S) \to B\bar{B}$ collision is needed.

6.1 Signal Event Selection

The process of selecting relevant events and to tighten the cut are separated into independent stages, with the following being a high level overview:

1. Start with the data provided by the central FEI hadronic skim, where generically decaying B, labeled as B_{tag} mesons are reconstructed.
2. In the second step, cuts on the beam constrained mass M_{bc} and the signal probability `sigProb` are applied.
3. In the third step, a signal–side B meson is reconstructed, labeled as B_{sig}. These are paired up with the B_{tag} to form an $\Upsilon(4S)$ to complete the complete decay string.
4. Finally, I analyze the rest of event (ROE) to reduce the number of charged tracks left in the decays.

This process requires reconstructing the signal–side B meson. I will explain the details of the methodology I have used next.

6.1.1 Signal Side Reconstruction

Efficiently identifying the signal–side B meson is a key step in any potential analysis of the decay of interest. In the case of the $B^0 \to K^{*0}\nu\bar{\nu}$ decay, there is only one single detectable daughter, the K^{*0}. However, it is necessary to start with the particles that show up in

Channel	Simulated Decay	Expected per ab^{-1}
Signal (K^{*0})	$B^0 \to K^{*0}\nu\bar{\nu}$	10.14×10^3
Signal (K_S^0)	$B^0 \to K_S^0\nu\bar{\nu}$	2.32×10^3
Mixed	$\Upsilon(4S) \to B^0\bar{B}^0$	534.6×10^6
uubar	$e^+e^- \to u\bar{u}$	1605×10^6
ddbar	$e^+e^- \to d\bar{d}$	401×10^6
ccbar	$e^+e^- \to c\bar{c}$	1329×10^6
ssbar	$e^+e^- \to s\bar{s}$	383×10^6

Table 6.1: Expected number of events for all decay channels of interest [27]

the final state of the complete decay chain. Here, the kaons themselves have two daughter particles that must be reconstructed in the initial step, namely the K^{*0} decays into charged pions and kaons $K^{*0} \to K^+\pi^-$. To reconstruct this subprocess, charged kaons and pions have to be identified in the final state. The software package `basf2` provides standard identification for a variety of particles, including charged kaons and pions. The cuts applied in identifying these particles are high purity lists for data studies:

- Check if `thetaInCDCAcceptance` is true. This variables returns `True`, if the particle is within CDC angular acceptance. It verifies if the particles' polar angle θ is within the range $[17°, 150°]$, while the angle is computed using the initial particle momentum.
- Require that at least 20 hits in the CDC can be associated to the track of the given particle.
- Require the tracks originate from close to the interaction point, $dr < 0.5$ and $|dz| < 2$ to be exact.

Finally, to distinguish between kaons and pions, the particle identification is set to a minimum. More specifically `kaonID` and `pionID` are required to be larger than 0.5. With the particle lists for the kaons and pions filled, I reconstruct the K^{0*} from the signal decay. I require the reconstructed K^{*0} to be within the 150MeV of the nominal K^{*0} mass. In practice this means that the mass should lie in the range $0.74\text{GeV} < M_{K^{*0}} < 1.04\text{GeV}$. From this reconstructed particle, the signal side B_{sig} is reconstructed. As the only remaining daughter particles are neutrinos, the kaon can be relabeled as the B_{sig} without applying any further cuts, due to the fact that in this study the invariant mass of the mother B is not of interest. With this, I was able to reconstructed the decay chain of the signal side.

6.1.2 $\Upsilon(4S)$ Reconstruction

The final step in the reconstruction is to produce an $\Upsilon(4S)$ candidate for each event. For this process I pair up every signal–side B meson candidate with every tag–side B candidate in a certain way to produce an $\Upsilon(4S)$. This pairing procedure guarantees that the paired signal and tag–side B mesons do not contain overlapping daughter particles. To combine the two B mesons, it has to be considered that the neutral B can oscillate (figure 6.1). It is important to take into account the phenomenon of B mesons changing (or oscillating) between their matter and antimatter forms before they decay. The B meson can exist as either a bound state of a strange antiquark and a bottom quark, or a strange quark and bottom antiquark. These oscillations are analogous to the phenomena that produce long and short–lived neutral kaons. Therefore, $\Upsilon(4S)$ candidates are reconstructed considering both B^0 and $\bar{B}^0$ as well as the $\bar{B}^0/\bar{B}^0$ and B^0/B^0 combinations.

In the last step, I analyze what remains in the detector, apart from the $\Upsilon(4S)$ candidate and its daughters. For the case that there are multiple $\Upsilon(4S)$ candidates in one event, I select the most likely and start to build the rest of event from here. To identify tracks in the ROE, several different requirements are applied:

- Limiting the maximal transverse momentum, which is momentum transverse to the particle beam in the accelerator: $p_t < 0.1$GeV
- Cut on the distance from the track to the decay origin: $dr < 2$cm and $|dz| < 4$cm.
- Require a minimum energy of the charged tracks: $E > 0.1$GeV to rule out any random noise.

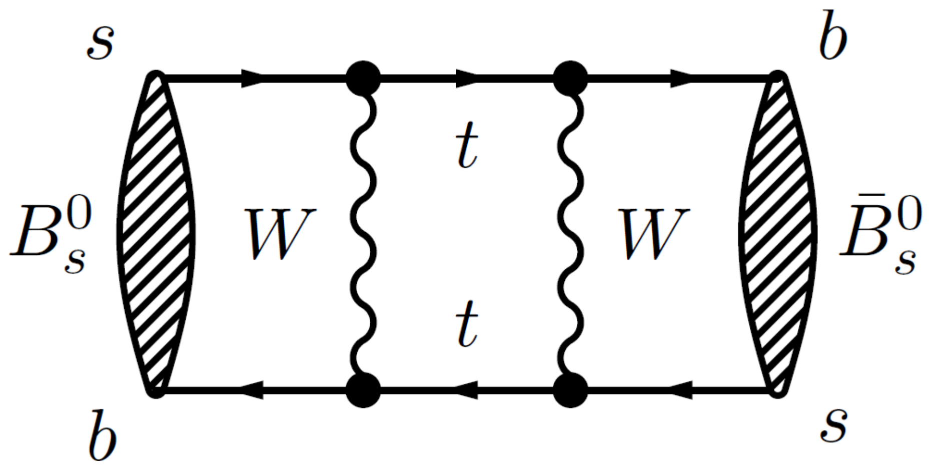


Figure 6.1: Feynman diagram of a neutral B meson oscillating between its matter and antimatter forms.

6.2 Results with Analysis Specific Filters

In this section I show the preliminary results for analysis specific filters for the smart background simulations. The general architecture of the Graph Attention Network is described in section 4.3. In addition, the speedup loss is used for all analyses. The previous

Model	GATGAP(gen)
Number of heads	4
Number of layers/modules	6
Number of units	32

Table 6.2: Optimal hyperparameters for the neural network filter for mixed background with no additional analysis specific filter.

study without any analysis specific filters found the optimal hyperparameters listed in table 6.2. I will use these parameters to first analyze the performance of the model with analysis specific filters for the mixed background. Next, I will introduce a streamlined way to search for the optimal hyperparameters for any analysis with tune [34]. Finally I will bring the optimal hyperparameters and cost–sensitive training data together to maximize the performance of the filter for both the mixed and quark continuum background.

6.2.1 Analysis Specific Filters for Mixed Events

To start off with, all models and results in this section have been trained with a balanced dataset of 1 Million pass and fail events each, except in the last case where I limit the tracks in the rest of event. The hyperparameters for the model are shown in table 6.2. When no analysis specific filter are specified and only the centrally commissioned FEI hadronic skim is considered, the speedup is limited (figure 6.2a). Figure 6.2b shows the speedup loss for training with 1 million events. The great attribute of the speedup loss is that it represents the inverse true speedup the filter would achieve in the application at the smart background Monte Carlo simulations. With a retention rate of 6.21% there are actually more training events available for this specific cut, which might marginally improve performance. The main limitation on the amount of events that can be loaded is the hardware, as more events mean longer training. However, only minimal performance benefits have been shown to be achieved with loading a significantly larger amount of additional data. Moreover, the training and validation losses are steadily decreasing until they reach a plateau. In addition, training and validation losses are correlated. This is an indication that the model is converging to its optimal state, using the data available during training. The final speedup that was reached with this setup is 1.84. This means that simulating the same effective sample size with these types of events is exactly 1.84 times faster with the NN filter than simulating them with the traditional brute force method

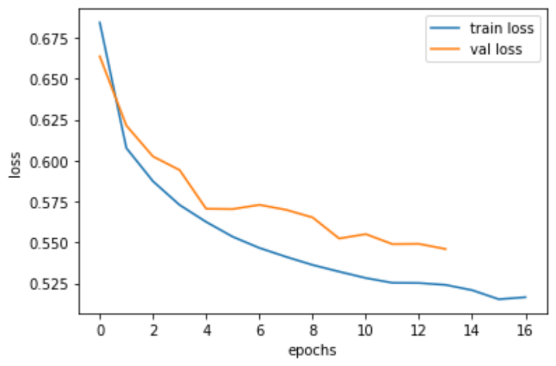
For the next cut, I increase the requirements for the beam constraint mass M_{bc} and the signal probability `sigProb`. This reduces the retention rate already by a significant amount to 2.28%. This cut is motivated by the fact that it increases the efficiencies on the reconstructed B_{tag}. Here, performance benefits are clearly shown. The speedup could be increased from 1.84 all the way up to 2.55, indicating already that analysis specific filters can be beneficial within the smart background simulation pipeline. Furthermore,

Cut

- **nTracks** < 13
- $M_{bc} > 5.24 GeV/c^2$
- $|\Delta E| < 0.2 GeV$
- **sigProb** > 0.001
- **nParticles(B0:feiHadronic)** > 0

retention rate $= 6.21\%$

(a) Regular FEI hadronic cut on the background Monte Carlo data. For more information, see chapter 4.1



(b) Training and validation loss per epoch with the speedup as loss function.

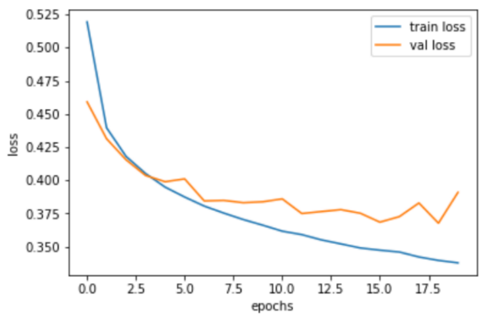
Figure 6.2: Performance of the model with no analysis specific filter, the final test speedup achieved with this selection is 1.84.

requiring that an $\Upsilon(4S)$ could be reconstructed with a $B \to K^{(*)}\nu\bar{\nu}$ decay is a good way to increase the performance of the filter, while at the same time increasing the efficiencies in the reconstruction of B_{tag}. This cut further decreases the retention rate by a factor of 5. The best Speedup achievable with this cut and these hyperparameters is 3.79.

However, reducing the retention rate has its limits. At some threshold, the cut is too tight and not enough events are left over for training. This is exactly the case for rejecting all events, where there are charged tracks left in the rest of event. The idea behind limiting tracks in the rest of events is that it ensures that all particles from the primary physics event were used in the $\Upsilon(4S)$ reconstruction. As can be seen in figure 6.4a, somewhat counterintuitively, the speedup is not at its minimum where the retention rate is the lowest. Instead the case of allowing 3 or less tracks maximizes the performance. Considering figure 6.4b it is easy to see why. There are not enough training events available to provide the model with enough statistics to properly learn the underlying structure of the data. Allowing exactly 3 or less tracks in the ROE is the optimal mix between reducing the retention rate and providing enough training data for this combination of hyperparameters. Working with weighted training data will improve this performance.

6.2.2 Hyperparameter Optimization

Within the topic of machine learning, hyperaparameter optimization is the process of choosing a set of optimal hyperparameters for a learning algorithm. Hyperparameters are parameters that are used to configure the learning algorithm. Common hyperparameters are the number of hidden layers, the number of neurons in each layer, the number of epochs, the learning rate, and the regularization parameters. As described in section 4.3, for the GATGAP model the main set of hyperparameters are the number of heads, layers and units. Due to the high complexity of the model used, training requires large amount of computational resources. One of the most used algorithms for hyperparameter optimization

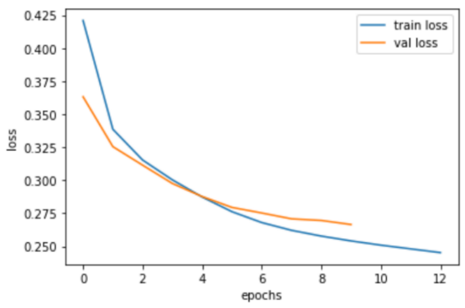


Cut

- ▶ **nTracks** < 13
- ▶ $M_{bc} > 5.265 GeV/c^2$
- ▶ $|\Delta E| < 0.2 GeV$
- ▶ **sigProb** > 0.005
- ▶ **nParticles(B0:feiHadronic)** > 0

retention rate $= 2.28\%$

(a) Results for cuts on M_{bc} and *sigProb* with the FEI hadronic skim, with Speedup = 2.55



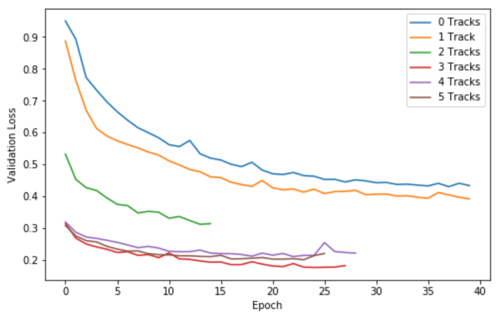
Cut

- ▶ **nTracks** < 13
- ▶ $M_{bc} > 5.265 GeV/c^2$
- ▶ $|\Delta E| < 0.2 GeV$
- ▶ **sigProb** > 0.005
- ▶ **nParticles(B0:feiHadronic)** > 0
- ▶ **nParticles(Upsilon(4S):reconstructed)** > 0

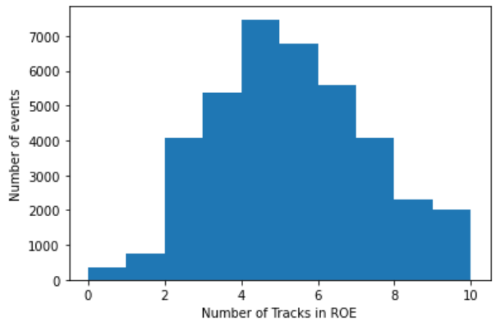
retention rate $= 0.45\%$

(b) Results for requiring that a $\Upsilon(4S)$ could be reconstructed with the decay chain $B^0 \to K^*\nu\nu$, with Speedup = 3.79

Figure 6.3: Effect of analysis specific cuts on the performance of the NN filter.



(a) Performance of the NN filter on a different number of tracks in the rest of event.



(b) Number of events per allowed tracks in the rest of event within a UDST file.

Figure 6.4: The struggle for finding enough statistics for the case of limiting tracks in the ROE and reducing the retention rate too much.

is grid search. This is a parameter sweep which performs an exhaustive search through a manually specified subset of the hyperparameter space of the model. This approach is not feasible especially for the case where multiple analysis specific filters are applied for a given physics analysis. Luckily, there are many more sophisticated methods for hyperparameter optimization, like Baysian or population based optimizations. However, in this particular research I have decided to focus on so called Tree of Parzen Estimators based distributed hyperparameter optimization [12]. The Tree–structured Parzen Estimator (TPE) is a sequential optimization approach, often referred to as sequential model–based optimization (SMBO). SMBO methods sequentially construct models to approximate the performance of hyperparameters based on historical measurements, and then subsequently choose new hyperparameters to test based on this model. Sequential in this scope refers to running trials one after another, each time trying to improve the performance of the previous trial by applying Bayesian reasoning and updating a probability model. The TPE approach models $P(x|y)$ and $P(y)$ where x is the set of hyperparameters and y represents the associated performance from past experiments. Using Bayes Rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}, \tag{6.1}$$

we try to model $P(y|x)$ from $P(x|y)$ and $P(y)$ and then use this distribution to find the best set of hyperparameters. The big advantage of SMBO is that no ranges for the hyperparameters have to be specified, instead the algorithm picks new sets of parameters on its own. This reduces the amount of input needed by the user making the framework more accessible for more physics analyses. In addition, experiments do not necessary have to be run sequentially, but multiple experiments can also be run in parallel. This reduces the total time needed for the hyperparameter optimization as long as the hardware is able to handle concurrent runs. Tune provides the possibility to use distributed cloud services with multiple GPUs.

Moreover, besides a sophisticated approach to find new parameters for running experiments, early stoppage is a useful technique to prevent overfitting and waste of computational resources. This can be implemented in two ways. First of all, a patience can be introduced in the training process of the model. Instead of defining a fixed number of epochs, after each epoch, the validation loss is computed. If the validation loss has not improved for a user defined number of epochs (the patience), training is aborted and the model parameters are set to values of the epoch with the lowest validation loss. Secondly, when working with sequential hyperparameter optimization, a scheduler can be added. These scheduled trials can be early terminated if they are labeled as bad trials. All Trial Schedulers take in a metric, in this research the validation loss, which is maximized or minimized according to the mode. If, after a minimum amount of epochs, the validation loss is much higher than for the best trial, the trial is considered bad and is terminated.

All in all, hyperparameter optimization with this setup is magnitudes faster and more efficient than the traditional methods, making it extremely useful if it is desired to apply multiple analysis specific filters for a specific physics analysis.

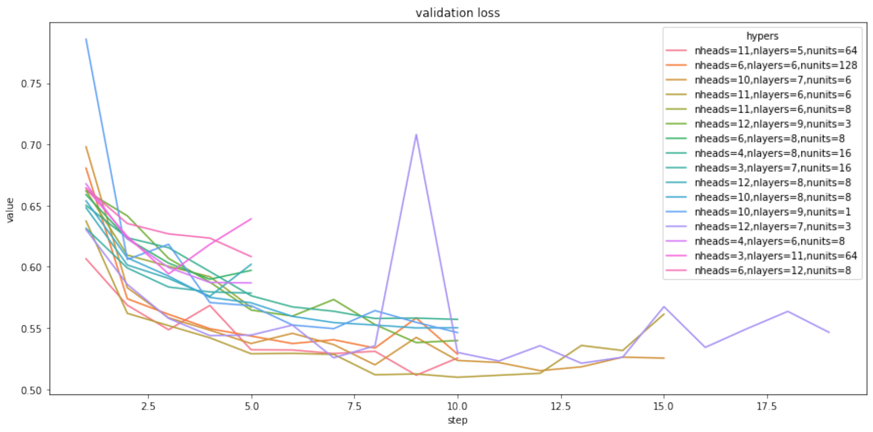
	Mixed Background	Continuum Background
Number of heads	9	11
Number of layers	8	5
Number of units	32	64

Table 6.3: Optimal set of hyperparameters for the GATGAP model.

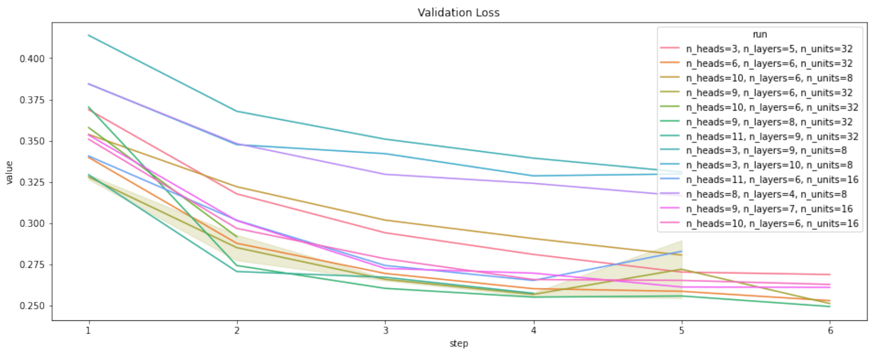
6.2.3 Results of Hyperparameter Optimization

There is a need to separately optimize the model for mixed and continuum background. For continuum background, the cuts introduced in section 6.1 lead to a higher reduction in the retention rates than for mixed background. For all experiments in this section an unweighted dataset was used. To reduce statistical fluctuations of the runs, I have used a large dataset with $1.5 \cdot 10^7$ pass and MDST fail events with additional 10^8 UDST fail events. Figure 6.5 shows the validation loss of the GATGAP model on mixed backgrounds for a variety of hyperparameter configurations and cuts. For the case where no analysis specific filter where applied (figure 6.5a), the best set of hyperparameters is 11 heads, 6 layers and 8 units. With this combination, the total speedup could be improved from 1.81 all the way up to 1.99. More importantly, figure 6.5b shows the validation loss of the GATGAP model on mixed backgrounds for a variety of hyperparameter configurations and cuts. For the case where analysis specific filter are applied. Here the best combination of hyperparameters is 9 heads, 8 layers and 32 units. Here an improvement from 3.79 to 4.01 could be achieved. It is evident that even for the same background, using different cuts for the analysis specific filters will lead to the fact that the optimal set of hyperparameters might change. Figure 6.5a clearly shows the scheduler at work as more epochs are needed for convergence. It is set to allow a minimum of 5 epochs to be allowed before the trial is considered bad. About 40% of all runs were bad, saving a lot of unnecessarily computations and reducing the need for more expensive Hardware. Analyzing the result for the runs with tight cuts for both continuum and mixed background in figure 6.5b closer, performance results can best be visualized with box–and scatterplots (figure 6.8). It is especially apparent that even for varying types of backgrounds, different hyperparameters are required to achieve optimal performance. The need to optimize the set hyperparameters for each cut and type of background makes it very important to tune the hyperparameters in this efficient way. The best set of hyperparameters of both backgrounds are summarized in table 6.3.

Due to the fact that very large training sets were used during training of both the continuum and mixed backgrounds, statistical fluctuations within runs with the same hyperparameters are expected to be minimal. Figure 6.7 compares the validation loss of all models to multiple runs of the best performing model. In addition, the test loss is shown for multiple runs of the best performing model. As expected the variance between runs is small. Even though this is not a guarantee that the model with the smallest validation loss is also the best performing model, it is a very good indication. Furthermore, it is expected that differences in performance between models with similar validation losses are only marginal and negligible for when the filter is deployed.

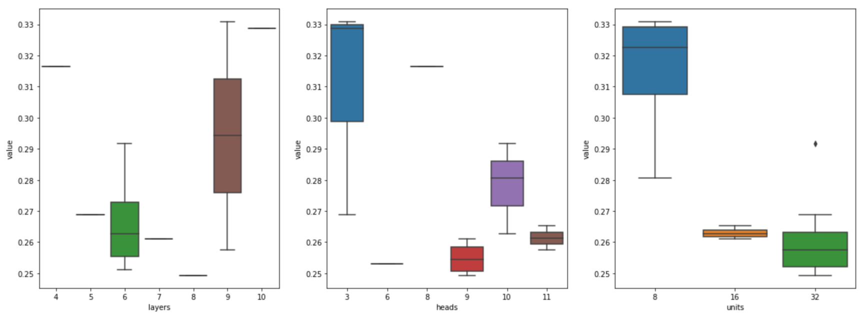


(a) Experiment runs for mixed background with the `feiHadronicB0` skim selection and no additional cuts.

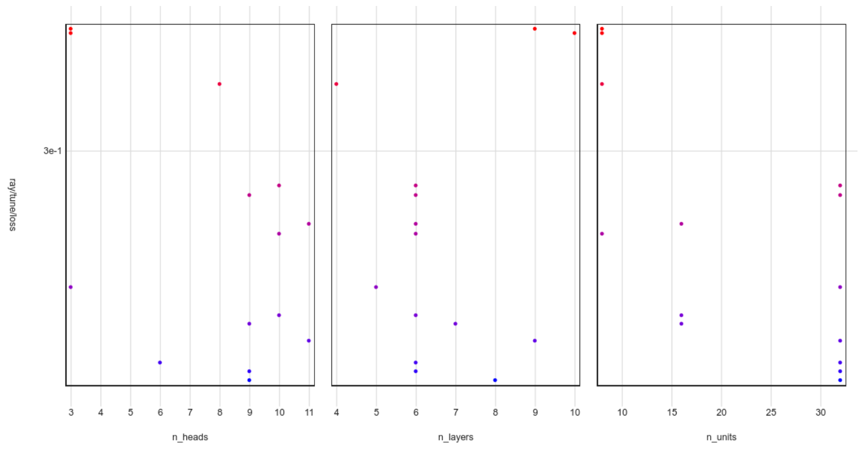


(b) Experiment runs for mixed background with analysis specific filter requiring a reconstructed $\Upsilon(4S)$ described in figure 6.3b.

Figure 6.5: Experiments with different hyperparameters for the case with no analysis specific filters and a tight cut. The runs that are shown are cleaned, failed runs with diverging loss have been removed



(a) Box plots of the performance of the hyperparamerts for mixed background.



(b) Scatter plot of the performance of different sets of hyperparameters for mixed background.

Figure 6.6: Results of the experimental runs for mixed background with analysis specific filter requiring a reconstructed $\Upsilon(4S)$ described in figure 6.3b.

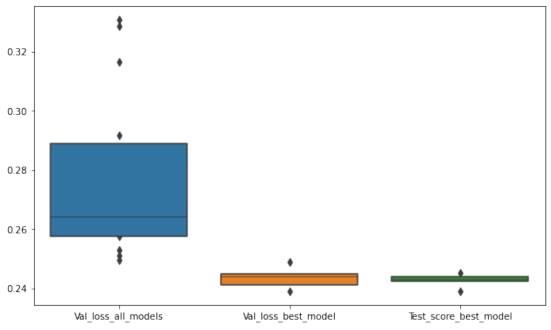
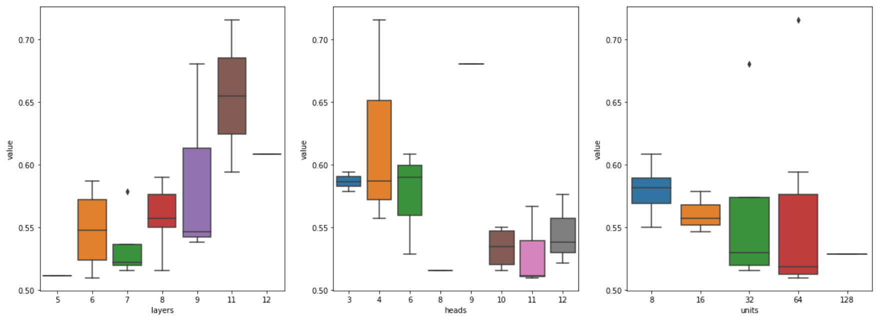
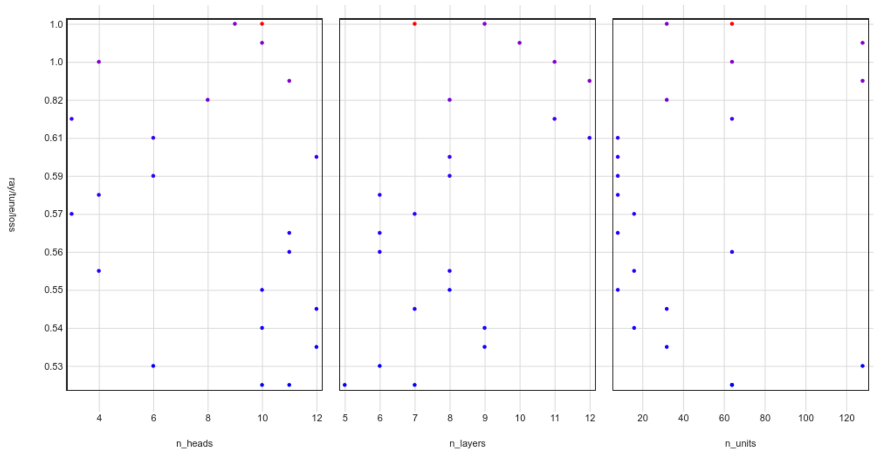


Figure 6.7: Variations between the validation loss of all models, multiple runs of the best performing model and the test loss of the best performing model.



(a) Box plots of the performance of the hyperparamerts for continuum background.



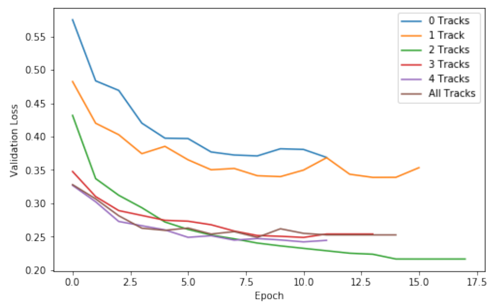
(b) Scatter plot of the performance of different sets of hyperparameters for continuum background.

Figure 6.8: Results of the experiemntal runs for continuum background with analysis specific filter requiring a reconstructed $\Upsilon(4S)$ described in figure 6.3b.

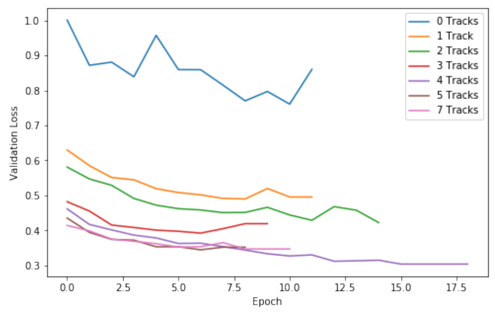
6.2.4 Performance of the Optimal Model on $B^0 \to K^{*0}\nu\bar{\nu}$

In this section I will give detail on the performance of the optimal model on the $B^0 \to K^{*0}\nu\bar{\nu}$ background. The complete framework of the smart background Monte Carlo simulation is expandable to any decay. In general, cuts with lower retention rates will improve the performance of the filter, as long as enough events are available to efficiently train the NN model. The models used in this section are initialized with the optimal set of hyperparameters for either background as shown in table 6.3. In addition, an unbalanced training set is utilized with the number of pass and MDST fail events limited to 10^7, while no more than 10^8 UDST fail events are loaded.

Comparing figure 6.9a, where an unbalanced dataset was utilized, to figure 6.4a, performance increases can be noticed. First of all, by using more failed events during training, the performance on allowing 3 tracks in the ROE slightly improved. Furthermore, the optimal combination of reducing the retention rate and available training events for training



(a) Performance of the optimal model on the mixed background.



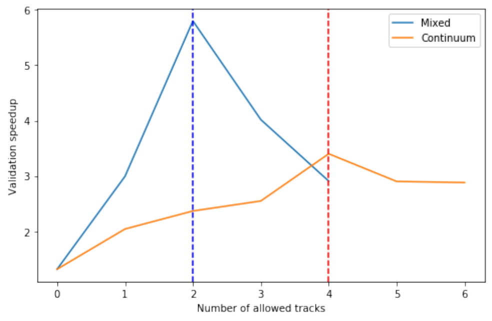
(b) Performance of the optimal model in continuum background.

Figure 6.9: Visualization of the validation loss per epoch for the optimal model using mixed and continuum background.

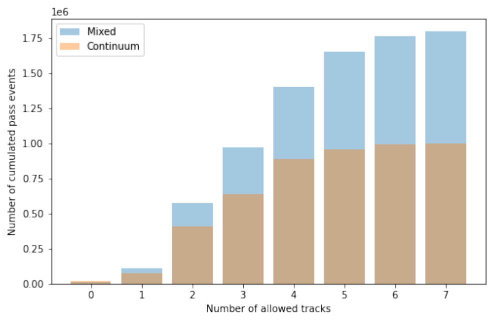
is achieved by allowing 2 tracks in the rest of event. The final speedup achieved with this cut on a test dataset is 5.81. It is a clear indication that in the case of using a balanced training set, not enough statistics is available to learn the underlying structure of the failed events. Using cost–sensitive training data can thus be very important in optimizing the speedup.

Considering the slightly lower retention rates for continuum background, more tracks in the ROE are needed for enough pass data to be available (see figure 6.10b). The optimal tracks in the ROE is 4 with a final speedup of 3.38. A visualization of the retention rate–sample size tradeoff can be seen in figure 6.10a.

As a final note, I have tried fintuning a pretrained model to increase performance with a tighter cut. Here, I have trained a model on a looser selection first with enough events available. In the next step a tighter selection is chosen and only the last fully connected layer of the network architecture is optimized. However, this has not yielded any performance improvements.



(a) Final speedup for mixed and continuum background with varying number of tracks in the ROE.



(b) Cumulative available training events for mixed and continuum background with varying number of tracks in the ROE. Each bar represents the number of available events with less or equal number of tracks in the ROE

Figure 6.10: Final speedup results for limiting tracks in the ROE combined with the available number of pass events at each selection.

Chapter 7

Summary

7.1 Results

In this work, I have followed the contributions of Kahn [27], Bross [13] and Yu [53]. To start of with, I have improved and streamlined the process of generating training data to be able to adapt the filter to improve simulation speeds for any physics analysis. I have added a new feature to the training data production to allow the user to specify analysis specific filters using `basf2`, including data from reconstructions using the most up to date Monte Carlo data available at time of writing this thesis. In addition, I have added cost–sensitive learning. Being able to feed a much increased number of data points to the NN during training can improve performance drastically. This is especially important in the search of rare decays with vanishing retention rates. Flavor changing neutral currents are extremely rare and sensitive to new physics, making this improvement essential for many interesting and important physics analyses. Furthermore, to further increase the performance, I have added an efficient and scalable procedure for hyperparameter optimization, reducing the need for expensive hardware significantly. Finally I have shown the potential of the filter for a specific physics analysis with the example $B^0 \to K^{*0}\nu\bar{\nu}$. Simulation speeds have improved from less than 2 in the case of no additional filters, all the way up to almost 6 for mixed background. Furthermore, with the development of the flexible training data production, different types of backgrounds can also be simulated using the filter. The best performance I was able to achieve for continuum background is around 3.4.

7.2 Outlook

The training data production and hyperparameter optimization with tune have been designed to be generalizable to any cut or network architecture. In addition, the cost–sensitive learning method is generalizable to data which is increasingly unbalanced, without loosing statistics on any minority class.

In the future, the extensions and improvements to the analysis specific smart background Monte Carlo simulation can be expected in the following directions:

- The graph neural network architecture can be improved upon, especially with the pace of research in graph neural network theory.
- Theoretical studies on the Speedup loss can be carried out. The loss might be able to be adjusted to improve the training process and subsequently lead to an increased the performance of the NN. Furthermore, finding a relationship between the speedup and cross–entropy can improve the performance by using both during training.
- With additional available luminosity in future Monte Carlo campaigns at Belle II, more data can be used for training. Especially for selections with low retention rates, this can drastically improve the performance of the NN.
- For extremely tight cuts, the NN can be trained on data from multiple Monte Carlo campaigns at the same time, careful validation on the actual performance on deployment would have to be conducted.
- Additional computational resources enable the possibility for more hyperparameter optimization to guarantee that the best set of parameters have been identified and the best performance has been achieved.

Bibliography

[1] *Belle 2 software documentation.*

[2] *SuperKEKB project.*

[3] A. Abashian, K. Gotow, N. Morgan, L. Piilonen, S. Schrenk, K. Abe, I. Adachi, J. Alexander, K. Aoki, S. Behari et al. , *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **479** (2002), 117.

[4] T. Abe, I. Adachi, K. Adamczyk, S. Ahn, H. Aihara, K. Akai, M. Aloi, L. Andricek, K. Aoki, Y. Arai et al. , *arXiv preprint arXiv:1011.0352* (2010).

[5] I. Adachi, T. Browder, P. Križan, S. Tanaka, Y. Ushiroda, B.I. Collaboration et al. , *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **907** (2018), 46.

[6] S. Agostinelli, J. Allison, K.a. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al. , *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **506** (2003), 250.

[7] K. Akai, K. Furukawa, H. Koiso et al. , *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **907** (2018), 188.

[8] J. Altmann, *Behaviour* **49** (1974), 227.

[9] D. Andrews, K. Berkelman, R. Cabenda, D. Cassel, J. DeWire, R. Ehrlich, T. Ferguson, T. Gentile, M. Gilchriese, B. Gittelman et al. , *Physical Review Letters* **45** (1980), 219.

[10] B. Aubert, A. Bazan, A. Boucham, D. Boutigny, I. De Bonis, J. Favier, J.M. Gaillard, A. Jeremie, Y. Karyotakis, T. Le Flour et al. , *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **479** (2002), 1.

[11] M. Beneke, G. Buchalla, M. Neubert und C.T. Sachrajda, *Nuclear Physics B* **606** (2001), 245.

[12] J. Bergstra, R. Bardenet, Y. Bengio und B. Kégl, *Advances in neural information processing systems* **24** (2011).

[13] Y. Bross: *Bias Mitigation in Selective Background Monte Carlo Simulation at Belle II.* Dissertation, 2020.

[14] O. Brüning, H. Burkhardt und S. Myers, *Progress in Particle and Nuclear Physics* **67** (2012), 705.

[15] A.J. Buras, J. Girrbach-Noe, C. Niehoff und D.M. Straub, *Journal of High Energy Physics* **2015** (2015), 1.

[16] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto und L. Zdeborová, *Reviews of Modern Physics* **91** (2019), 045002.

[17] H.Y. Cheng, *Physics Reports* **158** (1988), 1.

[18] P.T. De Boer, D.P. Kroese, S. Mannor und R.Y. Rubinstein, *Annals of operations research* **134** (2005), 19.

[19] M. Dine und A. Kusenko, *Reviews of Modern Physics* **76** (2003), 1.

[20] B.I.C. Education. *What is machine learning?*

[21] A.L. Fradkov, *IFAC-PapersOnLine* **53** (2020), 1385.

[22] E. Gabrielli, M. Heikinheimo, K. Kannike, A. Racioppi, M. Raidal und C. Spethmann, *Physical Review D* **89** (2014), 015017.

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

[24] K. Hornik, M. Stinchcombe und H. White, *Neural networks* **2** (1989), 359.

[25] T. Inami und C.S. Lim, *Progress of Theoretical Physics* **65** (1981), 297.

[26] G. Isidori und A. Retico, *Journal of High Energy Physics* **2001** (2001), 001.

[27] J. Kahn, E. Dorigatti, K. Lieret, A. Lindner und T. Kuhr: *Selective background Monte Carlo simulation at Belle II. Selective background Monte Carlo simulation at Belle II*, In *EPJ Web of Conferences*, Band 245. EDP Sciences (2020) Seite 02028.

[28] T. Kajita, *Reviews of Modern Physics* **88** (2016), 030501.

[29] T. Keck. In *Machine Learning at the Belle II Experiment.* Springer (2018), Seiten 63–100.

[30] M. Kikuchi, T. Abe, K. Egawa, H. Fukuma, K. Furukawa, N. Iida, H. Ikeda, T. Kamitani, K.i. Kanazawa, K. Ohmi et al. , *Proceedings of IPAC* **10** (2010), 1641.

[31] M. Kobayashi und T. Maskawa, *Progress of theoretical physics* **49** (1973), 652.

[32] T. Kuhr, C. Pulvermacher, M. Ritter, T. Hauth und N. Braun, *Computing and Software for Big Science* **3** (2019), 1.

[33] D.J. Lange, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **462** (2001), 152.

[34] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J.E. Gonzalez und I. Stoica, *arXiv preprint arXiv:1807.05118* (2018).

[35] L. Maiani, *arXiv preprint arXiv:1303.6154* (2013).

[36] D. Matvienko: *The Belle II experiment: status and physics program. The Belle II experiment: status and physics program*, In *EPJ Web of Conferences*, Band 191. EDP Sciences (2018) Seite 02010.

[37] R.K. Mitchell, B.R. Agle und D.J. Wood, *Academy of management review* **22** (1997), 853.

[38] M. Nielsen. *Neural Networks and Deep Learning[Internet]. Neuralnetworksanddeeplearning. com. 2020 [cited 11 July 2022].*

[39] Y. Ohnishi, T. Abe, T. Adachi, K. Akai, Y. Arimoto, K. Ebihara, K. Egawa, J. Flanagan, H. Fukuma, Y. Funakoshi et al. , *Progress of Theoretical and Experimental Physics* **2013** (2013).

[40] M. Pharr, W. Jakob und G. Humphreys: *Physically based rendering: From theory to implementation.* Morgan Kaufmann, 2016.

[41] A.S. Polydoros und L. Nalpantidis, *Journal of Intelligent & Robotic Systems* **86** (2017), 153.

[42] J.F. Ritchie Ng: *Deep Learning Wizard.* Zenodo, Apr 2019.

[43] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner und G. Monfardini, *IEEE transactions on neural networks* **20** (2008), 61.

[44] T.W. Shinn und T. Takaoka, *ACSC* (2013).

[45] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel et al. , *Science* **362** (2018), 1140.

[46] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen und P.Z. Skands, *Computer physics communications* **191** (2015), 159.

[47] S. Tapia und J. Zamora-Saá, *Nuclear Physics B* **952** (2020), 114936.

[48] N. Toge. *KEK B-Factory Design Report.* Technischer Bericht, 1995.

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser und I. Polosukhin, *Advances in neural information processing systems* **30** (2017).

[50] D.H. Wolpert, *Soft computing and industry* (2002), 25.

[51] R. Workman et al. , To be published (2022).

[52] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu und K. Weinberger: *Simplifying graph convolutional networks. Simplifying graph convolutional networks*, In *International conference on machine learning.* PMLR (2019) Seiten 6861–6871.

[53] B. Yu: *Improved Selective Background Monte Carlo Simulation at Belle II with Graph Attention Networks and Weighted Events.* Dissertation, 2021.

Acknowledgements

First and foremost, I want to thank Professor Thomas Kuhr who provided me with the opportunity to take part in this project. Without his valuable feedback on my findings and support during writing and presentations, this project would not have been possible.

Furthermore, I am extremely grateful to Dr. Nikolai Hartmann. With his deep knowledge in computer science and HEP and his willingness to pass on this knowledge, I could not have wished for a better supervisor. Not only did I learn a lot from him in the fields of physics and data science, but I was also able to gather valuable experience using Linux and distributed computational resources. I am thankful for all his ideas, support and careful correction of my thesis. Without him, my research would not have been possible.

In addition, I would like to thank all members of the AG–Kuhr group. From the beginning, everyone was happy to talk to me about their advanced research. The large variety of projects conducted in the group has greatly enriched my knowledge in and outside the field of HEP. I am especially grateful to Boyang Yu, who has help me get started in this project and supported me throughout my entire research.

Finally, I would like to thank my family for all their unconditional encouragement and support.

Erklärung/Declaration

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

I hereby declare that this thesis is my own work, and that I have not used any sources and aids other than those stated in the thesis.

München, 06.08.2022
Luca Schinnerl