Karlsruhe Institute of Technology

ETP-KA/2022_12

Improving Event Selection with Machine Learning Methods for the $B^{\pm} \to K^{\pm}a$, $a \to \gamma\gamma$ Search at Belle II

Frederik Schmitt

Bachelorthesis

7th October 2022

Institute of Experimental Particle Physics (ETP)

Advisor:	Prof. Dr. Torben Ferber
Coadvisor:	Dr. Slavomira Stefkova

Editing time: 1st July 2022 – 7th October 2022

Karlsruhe Institute of Technology

ETP-KA/2022_12

Verbesserung der Ereignis Auswahl mit Maschinellem Lernen für die Suche nach $B^{\pm} \to K^{\pm}a$, $a \to \gamma\gamma$ Zerfällen bei Belle II

Frederik Schmitt

Bachelorarbeit

7. Oktober 2022

Institut für Experimentelle Teilchenphysik (ETP)

Referent:	Prof. Dr. Torben Ferber
Korreferent:	Dr. Slavomira Stefkova

Bearbeitungszeit: 1. Juli 2022 – 7. Oktober 2022

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 7. Oktober 2022

. . . .

(**Frederik Schmitt**)

Disclaimer

This thesis uses Monte Carlo samples adopted from A. Heidelbach's (KIT, ETP) master thesis [1] and generated by him. Furthermore, the Python routines for plotting are adopted from him. The Punzi-net architecture used within this thesis is based on `https://github.com/feichtip/punzinet`. The idea to study a possible candidate selection with Punzi-net for the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ search at Belle II was proposed to me by T. Ferber (KIT, ETP) and A. Heidelbach (KIT, ETP). All of the analyses reported in this thesis are performed by me, all results are formulated by me, and all figures are my created by me unless otherwise noted.

Contents

1. Introduction

Physics as a science describing the phenomena of nature has dramatically evolved in the last one hundred years, answering more and more open questions. On the largest scale, the endeavours led to many successful theories and discoveries about stars, black holes, and the big bang. On the smallest scales, the perseverent search for the fundamental building blocks of nature discovered the atomic models with protons, neutrons, and electrons as well as effects like radioactivity questioning their indivisibility. Similar breakthroughs in electronics and computer science not only had a profound impact on everyday life but also allowed the opportunity to build large-scale experiments like particle colliders to dig deeper into the known particles and discover the elementary particles of the standard model. Although many predictions of the standard model are probed to incredible precision, physicists quickly noticed shortcomings in the theoretic description, which were incompatible with the experimental results, e. g., the neutrino's mass or the neutron's vanishing electric dipole moment.

These new open questions brought up different new physic theories to describe the effects beyond the Standard Model. One group of these theoretically predicted particles are the axion-like particles. The axion-like particles are a broad group of pseudoscalar particles with couplings to the standard model, with free parameters for mass coupling and mixing. This thesis considers a model in which the axion-like particle predominantly couples to the electro-weak sector of the standard model. Assuming the coupling to the W-boson being the strongest leads to an interesting signature in which the axion-like particle is produced in a one-loop flavour changing neutral current transition and decays into two photons. Consequently, arising signatures like $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ can be studied at high-intensity experiments like Belle II and BaBar.

Within this thesis, a neural network is used as a modern analysis technique to perform the signal selection on Monte Carlo events. To achieve this, the thesis adopts the previously implemented non-differentiable minimal detectable cross-section and a loss function for a neural network classifier based on this cross-section. The previous use of the so-called Punzi-net in the $e^+e^- \to \mu^+\mu^- Z'$ search [2] at Belle II showed that the novel Punzi-loss function can outperform standard multivariate analysis techniques. Within this thesis, the previous implementation is adapted for the search for the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ signature at Belle II. Therefore, several functions are changed to ensure compatibility with the

$B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ data set. Additionally, the thesis implemented loss validation to monitor the training process and improved output routines to obtain quantitative results. The training on the Punzi-loss was also improved to perform batched training for further optimization. This thesis shows that the Punzi-net approach to the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ analysis can attain results similar to the previous cut-based analysis. Furthermore, it shows that architectural improvements to the neural network can improve obtained results.

Chapter 2 introduces the experimental Setup of the Belle II detector at the SuperKEKB accelerator and discusses the main feature variables. Subsequently, Chapter 3 summarizes the principles and challenges of machine learning and neural networks. Chapter 4 describes the implementation of neural networks in the python programming language and especially the implementation of the Punzi-loss function introduced by [2]. Chapter 5 summarizes the results that this approach can achieve on the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ dataset in different setups.

2. The Belle II Detector and Variables used in *B* Physics

This chapter introduces the experimental setup of the SuperKEKB accelerator and the Belle II detector. Furthermore, it also discusses the methods and variables used to investigate the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ process.

2.1. The Belle II Detector

One way to measure the fundamental properties of elementary particles or search for new particles, e.g. axion-like particle (ALP), is using modern high-luminosity particle colliders and detectors. An example of such facilities is the Belle II detector at the SuperKEKB collider in Tsukuba, Japan. Fig. 2.1 shows the setup of this accelerator complex. SuperKEKB is an asymmetric electron-positron collider that operates mostly at a center-of-mass energy at the $\Upsilon(4S)$ resonance. This resonance decays predominantly into a pair of either charged or neutral B- and anti-B-mesons.

The B mesons subsequently decay through several modes. This thesis deploys improvements in the candidate selection of the search for the new physics signature of the ALP in the studied decay $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$.

Fig. 2.2 displays the Belle II detector, which consists of several subdetectors measuring specific quantities of different particles. Physical quantities like energy and momentum are reconstructed by combining the subdetector measurements.

Vertex detector and central drift chamber (CDC): The innermost sub-detectors measure the paths of the charged particles. At first, in the Vertex detector, semiconductor detectors measure the trajectories near the interaction point. The CDC continues the track measurement by measuring the ionization between multiple charged wires. Due to the magnetic field in the CDC, the tracks of charged particles curl into helices, carrying information about the particles' mass and charge [5].

Particle Identification: The particle identification sub-detectors Aerogel Ring Imaging Cherenkov Detector (ARICH) and Time of Propagation (TOP) use the Cherenkov effect to identify the particles by their masses. The Cherenkov cone's opening angle depends on

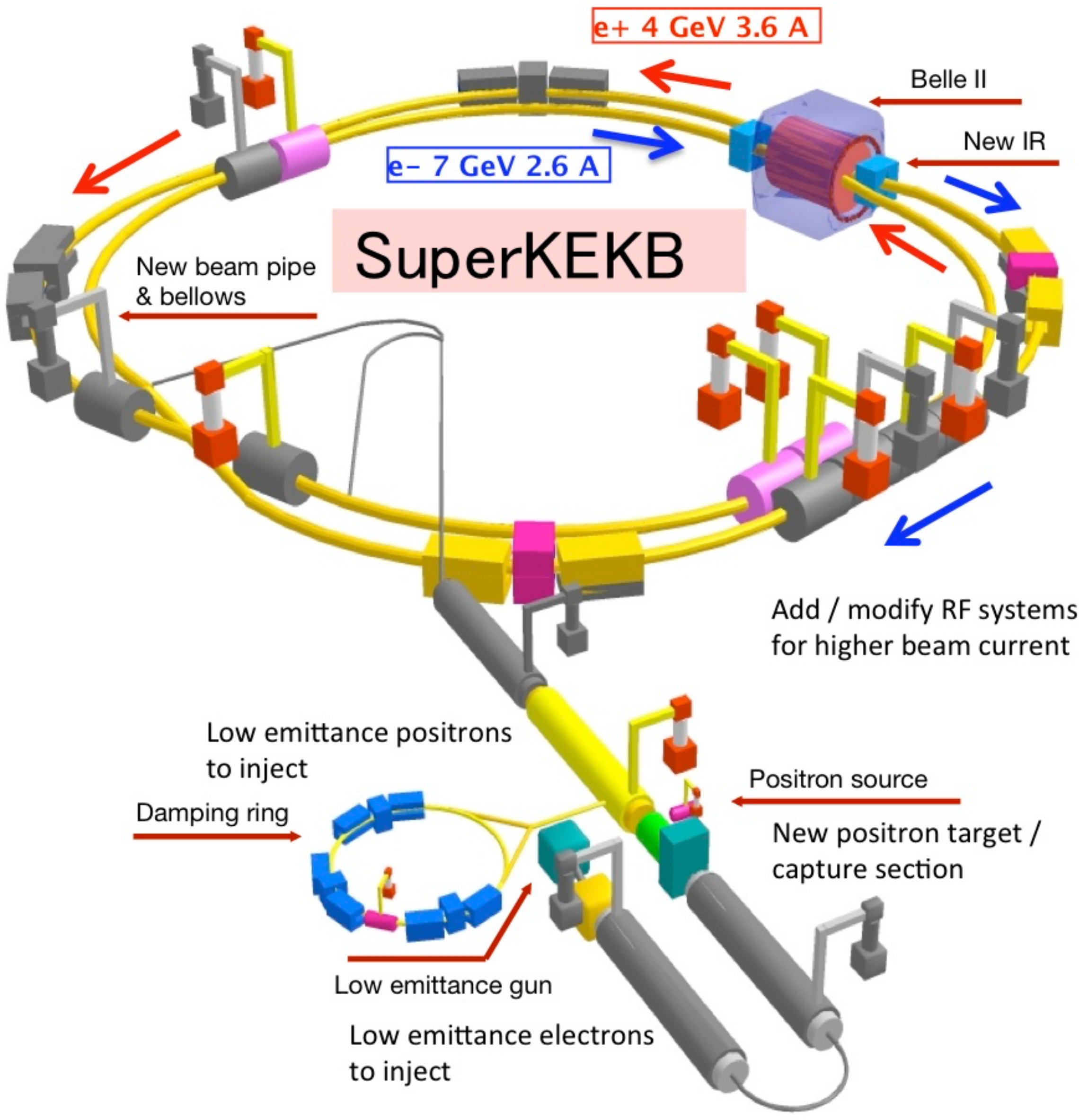

e+ 4 GeV 3.6 A
Belle II
New IR
e- 7 GeV 2.6 A
SuperKEKB
New beam pipe & bellows
Add / modify RF systems for higher beam current
Low emittance positrons to inject
Positron source
Damping ring
New positron target / capture section
Low emittance gun
Low emittance electrons to inject

Figure 2.1.: Schematic setup of the SuperKEKB accelerator. The electrons are injected with the electron gun at the bottom and accelerated in a linear accelerator. Positrons are created by the electrons hitting a target. Both beams are then injected into the storage rings. Adapted from [3].

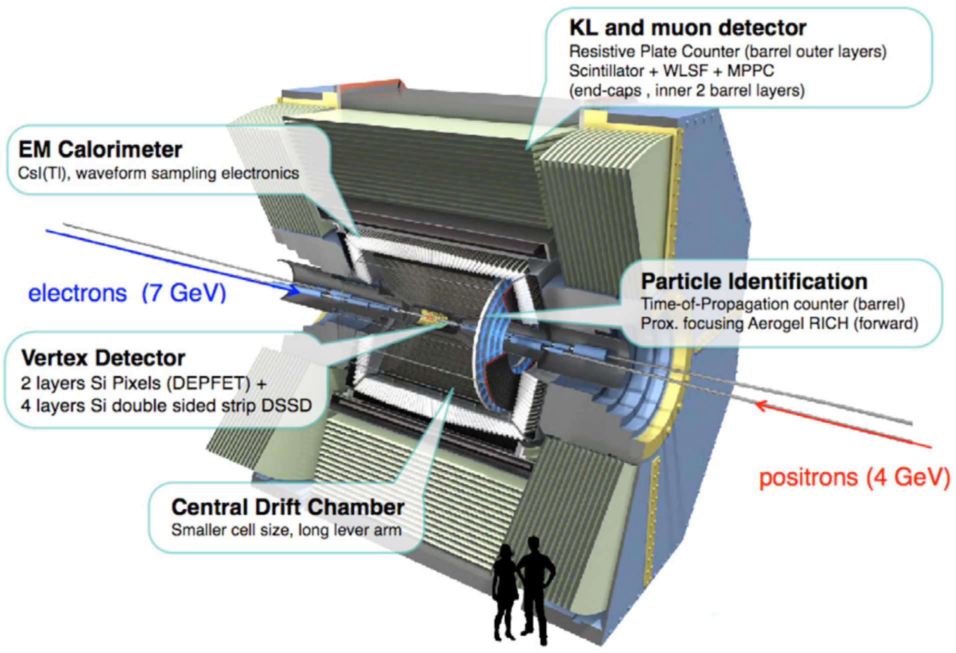

KL and muon detector
Resistive Plate Counter (barrel outer layers)
Scintillator + WLSF + MPPC
(end-caps , inner 2 barrel layers)
EM Calorimeter
CsI(Tl), waveform sampling electronics
electrons (7 GeV)
Particle Identification
Time-of-Propagation counter (barrel)
Prox. focusing Aerogel RICH (forward)
Vertex Detector
2 layers Si Pixels (DEPFET) +
4 layers Si double sided strip DSSD
positrons (4 GeV)
Central Drift Chamber
Smaller cell size, long lever arm

Figure 2.2.: Illustration of the Belle II detector and its subdetectors. This figure is adapted from [4].

the particle's velocity and refractive index n. Therefore, particles with equal momenta but different masses have different Cherenkov angles. ARICH measures the angle directly as the radius of the resulting ring image of the Cherenkov cone. TOP, on the other hand, uses multireflection of the Cherenkov light in thin plates to resolve the angle indirectly. Therefore detectors on the plates' edges read the time distribution of the incoming reflected light to resolve the total flight time and hence the opening angle.

Electromagnetic Calorimeter: The Electromagnetic Calorimeter (ECL) measures the energies of incoming particles via scintillator crystals. High energetic particles can excite electrons in a scintillator material which then return to their initial state while emitting lower energetic photons. As the emitted photons have low energies, photodiodes can measure their energies.

K_{L} and Muon Detectors: The outermost part of the detector is the K_{L} and Muon Detector (KLM), which uses an alternating structure of iron plate absorbers and resistive plate chambers and scintillator detectors to measure the energy loss of these long-lived particles.

These sub-detectors produce large amounts of raw measurement data, which are not directly usable. Elaborate reconstruction algorithms transform the raw data to physical quantities like momenta and energies. In the case of the Belle II experiment, these algorithms are part of the Belle analysis software framework (Basf2) [6]. This framework combines all methods for data unpacking, simulation, reconstruction, and analysis.

2.2. Monte Carlo Analysis

A critical principle in natural sciences is the neutrality of the results from the scientists' expectations. Meaning the experiment should be independent of any expectations. Especially in collider physics, the principle of a so-called blind-analysis is crucial as only a few facilities in the world could potentially reproduce or falsify results. Blind means that the complete analysis is first performed without looking at the actual signal. Collider physics typically tests the analysis on computer-generated data, also called Monte Carlo (MC) before the analysis switches to actual experimental data. To get these MC events, the desired events are simulated, converted into the detector's response using complex simulations and subsequently fed through the reconstruction algorithms.

As the production of these events is computationally expensive, the Belle II collaboration offers pre-produced samples for different standard processes in different experiment stages. This thesis considers continuum and generic B samples corresponding to $100\,\mathrm{fb}^{-1}$ of the early phase 3 detector geometry with corresponding background levels (2020-2022).

Continuum Background: SuperKEKB is tuned to produce as many B-mesons as possible with the center-of-mass energy at $\Upsilon(4S)$. Nevertheless, the majority of the events are non-B-events. Many of the unwanted events like Bhabha scattering or bremsstrahlung are already rejected by the triggers of the detector [5]. The remaining continuum processes for this analysis contain $e^+e^- \to q\bar{q}$ $(q = u, d, c, s)$ decays.

B-Samples: This category of Background contains all $\Upsilon(4S) \to B\bar{B}$ decays split into two categories. On the one side, the charged samples contain the $\Upsilon(4S) \to B^+B^-$ decays, and on the other side, the mixed samples consist of $\Upsilon(4S) \to B^0\bar{B}^0$.

The signal samples studied here are privately generated from the previous thesis considering ALP masses from $m_a = 0.175\,\mathrm{GeV}\,c^{-2}$ to $4.6\,\mathrm{GeV}\,c^{-2}$ with exclusions of the η and η' resonances [?] This work considers signal samples with an ALP mass above the η' resonance ($m_a > 1.01\,\mathrm{GeV}\,c^{-2}$) to avoid irreducible background. Furthermore, the training data set for the neural network (NN) contains only 30 of the masses from this range, leaving 4 masses for evaluation. This choice is motivated by testing the interpolation power of the trained classifier for untrained mass hypotheses.

2.3. Analysis Variables

From the reconstructed decay information, multiple physical quantities are calculated. A critical challenge in an analysis is to choose variables that are a good starting point to discriminate between signal and background events. Although modern multivariate analysis methods like NN can use minor differences and correlations in the signal and background distributions to find some representation of the data that separates signal from background, they possibly can not learn one from completely identical distributions. Furthermore, variables with different distributions for signal and background can provide a better starting point for the NN. An additional challenge comes with using too many features. As the dimension of the input space grows with each feature, significantly more samples are required for a higher number of variables. This "Curse of dimensionality" leads to keeping the input dimension small yet remaining the most discriminating power. Chapter 3, which focuses on NN and their use in particle physics, gives more details about the applied analysis techniques.

2.3.1. Kinematic Variables

The primary use of kinematic variables is to discriminate B events from continuum backgrounds. Two essential quantities for this are the beam-energy constrained mass M_{bc} and the energy difference ΔE [5].

The beam-energy constrained mass is defined by

$$M_{\mathrm{bc}} = \sqrt{{E^*}^2_{\mathrm{Beam}} - {\vec{p}^*}^2_{\mathrm{B}}}, \tag{2.1}$$

where E^*_{Beam} denotes the beam energy in the center-of-mass frame and $\vec{p}^*_{\mathrm{B}}$ the momentum vector of the B-meson candidate. Another commonly used variable is the energy difference

$$\Delta E = E^*_{\mathrm{B}} - E^*_{\mathrm{Beam}}. \tag{2.2}$$

These are the main quantities to discriminate correctly reconstructed B mesons from continuum background events and missreconstructed Bs. Other kinematic variables can be derived directly from the energies and momenta of the photon and kaon candidates:

- E_{Kaon} Kinetic energy of the Kaon candidate.
- p_{Kaon} Momentum of the Kaon candidate.
- $p^{\mathrm{T}}_{\mathrm{Kaon}}$ Transverse component of the kaon candidates momentum with respect to the forward beam direction.
- E_{γ^l} Energy of the lower energetic photon.
- E_{γ^h} Energy of the higher energetic photon.

Also, the combination of the photon energies and momenta gives the possibility to derive the invariant diphoton mass or center-of-mass energy $M_{\gamma\gamma}$. For correctly reconstructed photons originating from an ALP decay, this value should be equal to the ALP mass and have a peaking structure resulting from relativistic energy conservation. However, background events display an isotropic distribution of $M_{\gamma\gamma}$.

2.3.2. Topological Variables

Additional variables can be derived from the event topology in the detector. Firstly, $\cos\theta^{\mathrm{ROE}}_{\mathrm{B}}$ is the cosine of the angle between the B-meson candidates thrust axis and the rest of the

event (ROE) thrust. This quantity is uniformly distributed for B-events because the mesons are produced almost at rest. $q\bar{q}$ events, on the other hand, are produced with higher kinetic energies resulting in a jet-like structure of the whole event. This manifests in $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$ peaking at a value of one [5].
Another popular category of topological variables is the Fox-Wolfram-Moments H_l [7]. These moments combine the momenta $\vec{p_i}$ of each particle in an event with the angles between them $\theta_{i,j}$ to identify jet-like events. For a set of N particles, the l^{th}Fox-Wolfram moment is defined by

$$H_l = \sum_{i,j}^{N} = |\vec{p_i}| \cdot |\vec{p_j}| \cdot P_l(\cos\theta_{i,j}), \tag{2.3}$$

with the l^{th} Legendre polynomial [5]. This analysis exclusively uses the normalized second moment

$$R_2 = \frac{H_2}{H_0}, \tag{2.4}$$

as a variable to express event isotropy.

3. Neural Networks and their Use in Particle Physics

McCulloch and Pitts introduced the idea of NNs in 1943 [8]. Advancement in mathematical methods and computing power eventually brought their breakthrough. By now, neural networks perform data analysis and forecast tasks in many fields of science, economy, and society, which classical analysis methods or humans can not achieve.
This chapter introduces how NN especially fully connected feed-forward networks can achieve such performances. Furthermore, it describes possible challenges during the training and their solutions.

3.1. The Basic Unit: A Node

The most fundamental building block is a so-called node or neuron, which takes multiple input values and maps them onto a number [9].

To achieve this behavior, a neuron takes the inputs $X = (x_1, x_2, \dots x_m)$ and passes it through a linear function

$$u(w_1, w_2, \dots w_m, b) = \sum_{i=1}^{m} x_i w_i + b, \tag{3.1}$$

to obtain an intermediate output value u. In this calculation, the so-called weights w_i and biases b are not fixed a priori. For the actual output y, an non-linear activation function $f(u)$ is applied. There are multiple possible activation functions to choose from, which can later influence the learning performance. Fig. 3.1 shows the sigmoid and hyperbolic tangent activation functions used in Punzi-net. Where the sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}. \tag{3.2}$$

3.2. From the Neuron to the Net

NNs consist of many neurons linked together in a specific way. These connections improve generalization power and the ability to learn complex correlations.

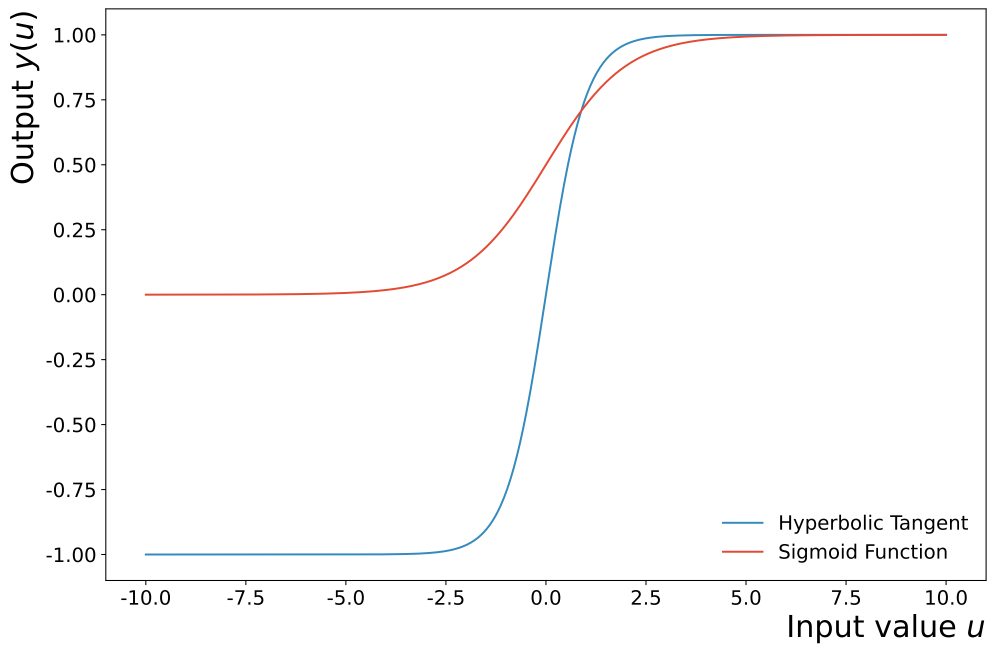


Figure 3.1.: Display of the shapes of the considered activation functions: hyperbolic tangent and sigmoid.

The first step toward the actual NN is to pass the output of a neuron as an input into another neuron. This would give a chain of two nodes in series. Each step in this serial structure is called a layer and the count of these layers is the depth [10].

Another possibility is to align multiple nodes in parallel to share the same inputs but with individual weights and biases, resulting in different outputs. This number of parallel nodes is also called layer width.

The combination of these two methods, building multiple layers with multiple nodes each, results in complex structures where several inputs are passed into the linear weighted sum and activated repeatedly. This nested structure is a so-called fully connected feed-forward NN which is a crucial part of this analysis.

To introduce the mathematical formalism of machine learning, the n neurons of one layer share the m-dimensional input vector $\vec{X} = (x_1, x_2, \dots x_m)^{\mathrm{T}}$, while their output is the n-dimensional vector

$$\vec{Y} = f\left(\hat{W}\vec{X} + \vec{b}\right). \tag{3.3}$$

With the weights in the shape of the matrix $\hat{W}$ and the biasses as $\vec{b}$.

The network mathematically performs multiple subsequent matrix multiplication and activation steps to compute the output value or vector. This subsequent processing of input data through the NN is called forward pass.

3.3. Training the Network

By now, the NN is still in a static state. To go the final step toward actual learning, the network needs some time evolution to upgrade itself. This thesis uses the so-called supervised training for learning.

To perform supervised training, it is mandatory to have not just a data set of multiple input variable vectors $\{\vec{X}_i\}$ but also the correct output values for these inputs $\{\tilde{y}_i\}$. One choice for the target values would be to assign the value of 1 to all signal samples and 0 to every background sample. The NN can now process an input vector $\vec{X}_i$ resulting in the network's predicted output y_i.

A so-called loss function provides a metric to compute the deviation to the desired output $\tilde{y}_i$. One reasonably common choice is the binary cross entropy loss (BCE), defined by

$$L = -(y\log(\tilde{y}) + (1-y)\log(1-\tilde{y})), \tag{3.4}$$

however, there are different functions applicable as loss functions. Chapter 4 presents a novel metric based on a function associated with the Punzi figure of merit as a loss function to outperform classical analyses [2]. Furthermore, additional weights, also called input weights W, adjust the influence of each sample on the loss calculation. These can correct for asymmetries in the data set, e. g. between the total number of signal and background events, to give each group the same weight during training.

Another way to formalize the loss function is as a function of the weight matrix $\hat{W}$ and bias vector $\vec{b}$ from Eq. (3.3) of each layer in the network. Respectively the output can be written as a function of the set of all weights and biasses, as well can the loss function

$$L(\hat{W}_1.\hat{W}_2, \dots \hat{W}_n, \vec{b}_1, \vec{b}_2, \dots \vec{b}_n). \tag{3.5}$$

In order to get the best possible result, the loss function has to be minimized with respect to each weight and bias. This minimization is analytically unsolvable as the number of parameters is too high and the exact shape of the loss function is unknown. Therefore, different numerical algorithms are used for this optimization. A straightforward example is the gradient descent algorithm, in which the gradient of the loss function is calculated for the given parameter point. [1] This gradient vector indicates the direction of the highest downhill slope in the loss curve with respect to the training parameters. The learning rate determines the step size of the optimization for the update step, which calculates the new parameters in the direction of the gradient vector.

Stochastic gradient descent (SGD) improves the computationally slow gradient descent algorithm by reducing the calculation steps. To do so, the data set is split into so-called batches and the gradient calculation and parameter updates are performed on each batch. One complete pass of the whole data set through optimization is also called an epoch. As the network should learn and improve its prediction, the loss value should fall in a usually exponential shape and converge if the learning is finished.

Optimization algorithms that use momentum and drag further improve SGD. These Momentum algorithms work like a ball rolling down a hill. Gradient changes are not entirely adapted in the optimization, but the algorithm maintains its momentum.

The main advantage of using momentum is that such algorithms are less likely to become trapped in local minima. A disadvantage of momentum-based algorithms are oscillations around the minimum value resulting in slower convergence. One of the implementations of momentum gradient descent is the adaptive moment estimation (ADAM) algorithm [11].

3.4. Hyperparameters and their Optimization

In the previous sections, many NN features have been mentioned but were not precisely quantified. Parameters like depth, width, learning rate, or the particular choice of the optimizer and activation functions have one thing in common they are arbitrary and untrainable. As these values can significantly influence a network's performance, they are also referred to as hyperparameters.

In practice, the effect of the hyperparameters on the training is large: they can change the behaviour of the network from training in seconds to training in hours; from finding an excellent solution to not learning at all. Therefore, finding a proper set of hyperparameters suitable for the NN task is crucial. This search can be done by hand or by special programs.

[1]Mathematicaly the gradient $\nabla f(\vec{x})$ shows the direction of steepest rise for a given point. In terms of NNs, it is common to invert the gradient definition to show the direction of the steepest descent.

Programs like Optuna perform this optimization more or less automatically by scanning the hyperparameter space for the best solution [12].

As these programs have their own so-called second-degree hyperparameters, like which parameters they should scan or in which value range they should search for each parameter, they must be used carefully. Furthermore, there comes a tough decision between the time consumption of the search on the one hand and the bias that it might miss the optimal set by keeping the scan space too small on the other hand. Especially since the network needs to train one or multiple times for each checked hyperparameter setup, the search can become quite time-consuming for larger numbers or parameters.

3.5. Training Challenges

As implied multiple times in the previous sections, training a NN can fail in different ways. A major challenge is the so-called overtraining or overfitting, which occurs when the network trains too long or is too complex. After the actual differences have been learned, the network starts to learn differences specific to the training data set. Therefore the network will perform worse if challenged to evaluate other data. A common monitoring technique is the so-called validation loss which calculates the loss on an untrained data set. The validation data set is disjoint from the training set but from the same underlying distribution. If overtraining occurs, the loss on the training data set falls or stagnates while the validation loss rises. To cope with overtraining in cases where the training loss already shows a good convergence, it is usually enough to stop the training earlier, which means reducing the number of epochs. However, if overtraining occurs very early, the hyperparameter setup has to be tested again.

Another challenge is that the NN does not train at all. This could occur in two cases. Firstly, the network learns much too slowly, resulting in an early stagnation of the loss. The network could train at an increased learning rate. Secondly, it is also possible that the network architecture is unsuitable for learning the difference between the input features, which requires an improved architecture. Thirdly, input features show too few differences for the network to find a reasonable classification.

A network that fails to train could also originate from a too high learning rate. As the learning rate indicates the step size in the optimization, the network is not optimized to its optimum configuration in this case. Instead, the optimization oscillates around the minimum. This can be solved by using a lower learning rate or a so-called learning rate scheduler. The scheduler changes the learning rate at a given condition. One example could be to reduce the learning rate by a factor if the loss does not reduce for a given number of epochs.

4. Implementation of the Punzi-Loss Function into the ALP Search

This thesis aims to implement the minimal detectable cross-section from Eq. (4.6) as a loss function in a NN to perform the signal extraction in the search for $B^{\pm} \to K^{\pm}a$, $a \to \gamma\gamma$. To do so, the work uses the Punzi-net implementation [2]. This Chapter explains the Punzi-net implementation for the $e^{+}e^{-} \to \mu^{+}\mu^{-}Z'$ search, as well as the basics of implementing NNs with `PyTorch`.

4.1. Minimal Detectable Cross-Section and Punzi Figure of Merit

Figures of merit (FOMs) are functions used to quantify performances. e. g. a metric for the performance of a classifier, like a NN output, which assigns signal and background events to values between zero and one. In this case, a figure of merit can calculate the optimal selection keeping as much signal as possible but also rejecting most background.
One realization is the Punzi figure of merit which is derived by the minimal detectable cross-section of a Poisson distributed process.
In statistical analysis, hypothesis tests are a commonly used method to make probabilistic statements about probed aspects of a given distribution. These tests are used in multiple scientific fields, e. g. checking the psychosocial influence of the Covid-19 pandemic [13] or if the observed excess in the background distribution is significant enough to be considered a newly detected particle.
For a new particle search, the hypotheses to test are, on the one hand, the zero hypothesis H_0, which represents the absence of signal events. On the other hand, the test hypothesis H_m that S_m signal counts above the background level B are present for a given set of parameters m. The Poisson probability density functions of these hypotheses are [14]

$$\begin{aligned} p(n|H_0) &= \frac{e^{-B}B^{n}}{n!}, \\ p(n|H_m) &= \frac{e^{-B-S_m}(B+S_m)^{n}}{n!}. \end{aligned} \tag{4.1}$$

From these distributions, the required number of signal events S_{min} to accept the signal hypothesis at a given confidence level can be expressed as [14]

$$S_{\min} = a\sqrt{B} + b\sqrt{B+S_{\min}}. \tag{4.2}$$

To get the actual needed minimal number of signal counts, we can solve the equation for S_{min}

$$S_{\text{min}} = \frac{b^2}{2} + a\sqrt{B} + \frac{b}{2}\sqrt{b^2 + 4a\sqrt{B} + 4B}. \tag{4.3}$$

The further introduction of the detected signal counts as a function of the integrated luminosity L, cross-section σ and signal efficiency ϵ via $S_{\text{min}} = \epsilon L \sigma_{\text{min}}$ leads to the term for the minimal detectable cross-section

$$\sigma_{\text{min}} = \frac{\frac{b^2}{2} + a\sqrt{B} + \frac{b}{2}\sqrt{b^2 + 4a\sqrt{B} + 4B}}{\epsilon L}. \tag{4.4}$$

The desired selection should minimize the detectable cross-section with respect to the selection. This minimum is the Punzi figure of merit

$$PFOM = \frac{\epsilon}{\frac{a}{2} + \sqrt{B}}. \tag{4.5}$$

To obtain the optimal selection, one usually must maximize this equation concerning the selection variable and only take what is left on either the right- or left-handed side of the selection.
In the derivation of Eq. (4.2), only the first order of the gaussian approximation is considered. Extending the calculation to the following order is possible to adapt the asymmetries in the Poisson tails better. This correction leads to a slightly different cross section [14]

$$\sigma_{\text{min}}^{\text{corr.}} = \frac{\frac{a^2}{8} + \frac{9b^2}{13} + a\sqrt{B} + \frac{b}{2}\sqrt{b^2 + 4a\sqrt{B} + 4B}}{\epsilon L}. \tag{4.6}$$

4.2. Neural Networks in `Python`

As multiple sectors of industry or research commonly use the concept of NNs for many data science tasks, different packages implement the general processes for machine learning described in Chapter 3 for many programming languages. One standard and recent implementation developed by a subgroup of Facebook is `PyTorch` [15]. `PyTorch` strongly focuses on creating a powerful yet easy-to-use framework for broad machine learning applications. As this package is also used for signal extraction in this thesis, the methods to implement a NN are briefly summarized.
The basic workflow for machine learning in `PyTorch` consists of the following steps according to Chapter 3. [1]

1. Definition of the model as a python class. This already fixes the hyperparameters for width, depth, and activation functions. By inheriting from the build in `torch.nn.Module` class, `PyTorch` can transfer many common routines automatically onto the new NN class. To be the starting point for the NN workflow, the class only needs an `__init__` method defining the shape of the architecture, and a `forward(x)` method. The forward command processes the inputs `x` through the network's layers and activation functions.

[1]This brief history of a `PyTorch` workflow is not meant as a complete tutorial. For beginners, the website of the `PyTorch` team offers tutorials following clear and practical examples under https://pytorch.org/tutorials/.

2. Preparing the data to be readable for `PyTorch`. Input data and target arrays typically are `NumPy` [16] arrays. As `PyTorch` can not read this datatype, they are converted into torch tensor with the `torch.from_numpy` command. `PyTorch` tensors also offer essential functionalities for machine learning, like specifying the device used for calculations. Therefore, the tensors can be moved to a specific device via `.to(device)`, which will most commonly either be CPU or GPU.

3. Initialize the desired loss function and optimizer. Many of which are already implemented to `PyTorch`'s `torch.optim` classes.

4. Perform the actual training loops over epochs and, if desired, batches as plain `Python` `for`-loops. Before each output calculation, the saved gradients of the network have to be reset with `optimizer.zero_grad`. Then the networks output for the input tensor `X` can be calculated via the intuitive `net(X)` call. The calculation of the loss value for a batch or epoch is similarly intuitive, which then can perform the gradient calculation and backpropagation with the `tensor.backward()` method and stores the gradients internally. To perform the actual update step of the parameters based on the gradients and the optimizer defined in the beginning, `optimizer.step` is used.

5. For Using the trained model, the evaluation can be done equally to the one in the training loop, but as no gradients are necessary anymore, the `torch.nograd` and `tensor.detach` methods must be used. Furthermore, `PyTorch` offers the ability to save and load the trained network parameters, as training can be time-consuming or results must be repeatable.

These steps summarize the workflow to implement NNs with `PyTorch` as a starting point to understand the following implementation of Punzi-net event selection.

4.3. Previous Implementation of Punzi-loss in the $e^+e^- \to \mu^+\mu^- Z'$ Search

The previous work [2] implements the minimal detectable cross section as a loss function for a NN. This network, called Punzi-net, performs the event classification in the $e^+e^- \to \mu^+\mu^- Z'$ search at Belle II. The resulting algorithm is implemented into python and `PyTorch` as the Punzi-net package. [2]. The package provides a complete toolkit for data preparation, pretraining on the BCE-loss, final training on the Punzi-loss, and analysis tools.
Besides the `PyTorch` workflow for training described in Section 4.2, the package offers several functions grouped into four sub-packages.

4.3.1. Preparation Functions

The preparation section contains all necessary functions for data and network preparation. The `set_weights()` function sets the input weights for the NN training. These weights are used in the loss calculation to adjust how much influence each sample has on the calculations. For the three simulated background types $\tau^+\tau^-$, $\mu^+\mu^-$, and $e^+e^-\mu^+\mu^-$ the background is

[2]The code of Punzi-net is open source and can be found along with some examples under https://github.com/feichtip/punzinet.

weighted with the inverse integrated luminosity by type. The signal is then weighted such that the sum over the signal weights equals that of the background. The function adds these values to the event variables, which are stored in a pandas DataFrame [17] object which is a particular file type to store and process more extensive data.

Furthermore, the `bin_widths` an `set_range_index` functions allocate the events to mass bins around the generated Z' masses by the reconstructed mass. For each event, the function stores the lowest and highest bin index in which the event occurs in the data frame. During this, the events occurring in zero bins, equal to events outside the signal range, are removed from the data set.

4.3.2. Functions Associated with the Punzi Figure of Merit

The first function of the figure of merit subclass `gen_sparse_matrices` processes the bin indices from above into matrices to later calculate the signal efficiency and background events. These indices are then converted into matrices with the dimension length of the data set times the number of mass bins. To encode if the event in column i can be detected in mass bins j, the matrix contains a one at position (i, j). Vice versa, each entry for a bin in which the event is not detected contains zero. In total, the function returns two matrices, one containing the signal events and one with background only. These matrices are filled mainly with zeros and only a few ones. For these so-called sparse matrices, it is better to store only nonzero entries and their positions to avoid flooding the memory with thousands of zeros. Different `Python` packages like SciPy [18] already offer dedicated classes for this purpose.

The actual `punziloss` function then calculates the loss value, using the sparse matrices for signal $\hat{\sigma}_S$ and background $\hat{\sigma}_B$ from above. To calculate the loss according to Eq. (4.4), the matrices must be combined with the networks output vector $\vec{Y}$ and inputs weights $\vec{w}$ to get the signal efficiency $\vec{\epsilon}$ and background events $\vec{B}$ for each trained mass. They are implemented as

$$\begin{aligned} \vec{\epsilon} &= \frac{\hat{\sigma}_S \cdot \vec{Y}}{n_S}, \\ \vec{B} &= \hat{\sigma}_B \cdot (\vec{Y} \circ \vec{w}), \end{aligned} \tag{4.7}$$

where $\circ$ denotes the elementwise or Hadamard product defined by $(\vec{u} \circ \vec{v})_i = u_i v_i$ and $\cdot$ is the common matrix multiplication. This way of implementing the signal efficiencies and background events as weighted sums has the advantage of being differentiable concerning the network parameters, which is necessary for backpropagation [2].
Based on signal efficiency and background events, the loss is calculated using the corrected minimal detectable cross section from Eq. (4.6) via the `punzisensitivity` function.

In addition, two functions for applying selections are provided. `fixed_cut` selects at a fixed given output value. Secondly, the `optimal_cut` function performs the selection which achieves the maximum Punzi figure of merit (Eq. (4.5)).

4.3.3. Training Functions

The training subclass contains the methods to perform the two training routines. At first, the `bce-training` function which performs the BCE pretraining according to Section 4.2.

Secondly, the `punzi-training` function performs the actual training on the minimal detectable cross-section loss, which was implemented above. With this change, the procedure also follows the workflow described in Section 4.2. To visualize the performance, the output classifier for signal and background can be plotted against the mass in two-dimensional histograms.

4.4. Deploying Punzi-net into the Axion-Like Particle Analysis

This thesis extends the existing and previously mentioned Punzi-net code to achieve compatibility with existing data frames from the ALP analysis. Additionally, several functionalities to monitor the training process and evaluate output performance are added to the workflow.

4.4.1. Data Preparation

The previous ALP analysis provides MC samples split into different files for the background and each generated mass. Furthermore, the MC events do not contain a candidate selection, meaning that each event has multiple candidates. For the sake of simplicity, this work applies a random candidate selection. In the random candidate selections, for each unique index and event type, one event is randomly chosen. The candidate selection involves iterating through all event indices, which makes the process time costly. Therefore, the candidate selection is performed separately and the results are saved to files.

The `load_df` and `read_edges` functions where added to the `punzinet` package to read the candidate selected samples for specified masses. To determine the mass bins, this analysis uses quantiles of double-sided crystal ball fits in $M_{\gamma\gamma}$. These bins are calculated by integrating the fit function from the mean value to the right. The $M_{\gamma\gamma}$ value at which the integral contains 99 % of the total area on this side is used as the selection. The left-handed selection is calculated similarly [1]. As the double-sided crystal ball function is asymmetric, the resulting $M_{\gamma\gamma}$ selection for each mass bin is also asymmetric. This is implemented in the changed `set_range_index` function.

Furthermore, the input data is reweighted to account for the background processes' different cross-sections σ. The resulting weights for the background type i are calculated by

$$w_i^{\text{Bkg}} = \frac{L_T}{L_i \sigma_i}. \tag{4.8}$$

The signal weights for each ALP mass m are also changed to account for different reconstruction efficiencies using the generated number of events N_{Gen} and the reconstructed number N_m to

$$w_m^{\text{Sig}} = \frac{N_{\text{Gen}}}{N_m}. \tag{4.9}$$

Furthermore, the signal weights are rescaled such that the sum of the signal weights is equal to those of the background weights.

4.4.2. Tailor the Training Routines

As described in Chapter 3, a validation loss is a commonly used tool to monitor obvious overtraining. The implementation of a validation loss in the BCE pretraining is similar to the regular procedure described in Section 4.2. The validation loss is calculated at the end of each epoch for the complete validation data set.

For the Punzi-training, the validation loss is more challenging than for the pretraining, mainly since the background event term depends strongly on the data set length. Many efforts to correct this issue, e. g. the introduction of scaling factors or the change of input weights for different data sets, do not achieve reasonable behaviour. Finally, a batched training and validation for Punzi-loss introduces a common norm factor, as the same batch size can be used for both calculations.
Batching the data for each epoch (as in the BCE training) would require calculating the sparse matrices for signal and background again, which is a time costly computation. Therefore the batches in the Punzi-training are chosen to be static to save runtime by only calculating the matrices once. This procedure is motivated in the Punzi-net paper, which also presents batched Punzi-training [2]. [3]

4.4.3. Output and Evaluation Functions

Some additional functions are implemented to better evaluate the NN's output compared to the two-dimensional histograms offered by Punzi-net. First, a one-dimensional histogram is added for visualization of the output distribution. Although this version can not show output differences through the mass range, the one-dimensional case is preferred as it looks much clearer and is easier to understand. Furthermore, in the one-dimensional version, the background types are split to visualize better which of the different backgrounds is challenging for the network.

The actual shapes of the Punzi figure of merit curves from Eq. (4.5) or other figures of merits can be calculated via the `PunziCuts` function. In difference to the Punzi-net implementation for signal efficiency and background events, this function uses a more straightforward implementation of these quantities, which aims to compare with [1]. To calculate the signal efficiency, the signal events surviving the selection for a given mass hypothesis are divided by the generated events for this hypothesis. An additional factor

[3] The code to perform this training is not publicly available via the GitHub repository. Therefore we contacted the corresponding author Paul Feichtinger, who provided us with the actual Punzi-training function, which was then adapted further.

corrects for the fact that due to the division into training and validation data sets, each set should contain less generated signal. This procedure then gives the percentage of detected signal events with respect to the generated number. Additionally, background events for each hypothesis are calculated as the number of remaining non-ALP events after the selection in the corresponding mass window.
The function also calculates the optimal selection based on the maximum figure of merit value or for a fixed output value and the total remaining background events and signal efficiency resulting from that selection. Fixed selections are necessary when the intermediate region of the output classifier is almost completely depleted and most events are classified as zero or one. In these cases, the figures of merit offer no meaningful selection value as their curves are mostly dominated by statistical fluctuations of the output in the intermediate regions. Therefore it is necessary to perform a selection at a specified output position, whose numerical value has only a minor effect on the resulting values due to the low population.

5. Implementation Results

This chapter summarizes the performances Punzi-net achieves and compares them with the previous (cut-based) analysis method in [1].

It first presents the results of training two different architectures on the initial variables used in the previous analysis. This approach studies if Punzi-net can achieve similar signal efficiencies and background rejections as the previous analysis and if improved architectures and hyperparameters can improve the performance.

Secondly, it presents the training on an additional set of input features. This set leaves the variables M_{bc} and ΔE out of the training to use them for fitting in the subsequent analysis.

5.1. Training on the Initial Set of Features

The main branch of this work uses the same variables as in the previous analysis, shown in Table 5.1 [1].

The expectation is that the variation between kinematic and topological variables can greatly suppress the continuum background. B background, on the other hand, is expected to be more challenging as only the energy of the lower energetic photon E_{γ^l} shows separation.

5.1.1. Input Feature Distributions

Histograms of each input feature for different masses visualize the initial separation of the signal and different backgrounds. As a compromise between the number of plots

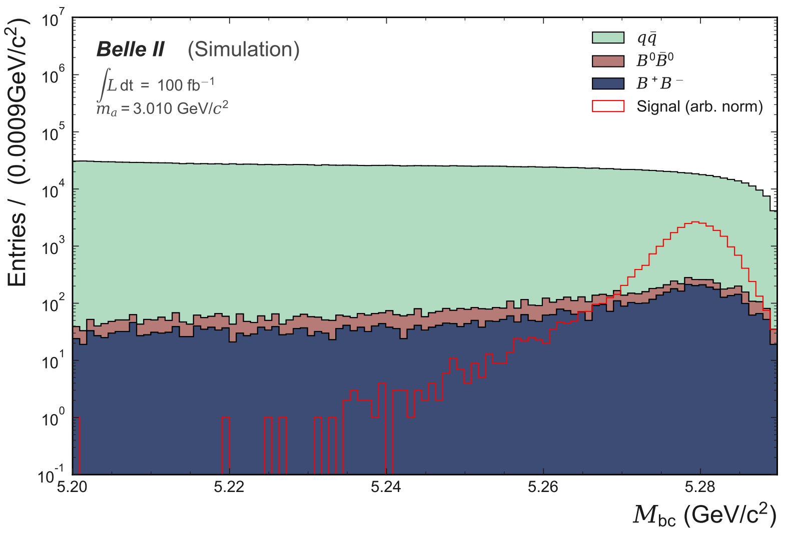
Table 5.1.: The main variables used for training Punzi-net. This variable set was adapted from the previous cut-based analysis [1] to compare with the Punzi-net directly.

Feature	Variable Name
1	M_{bc}
2	ΔE
3	E_{γ^l}
4	$\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$
5	R_2

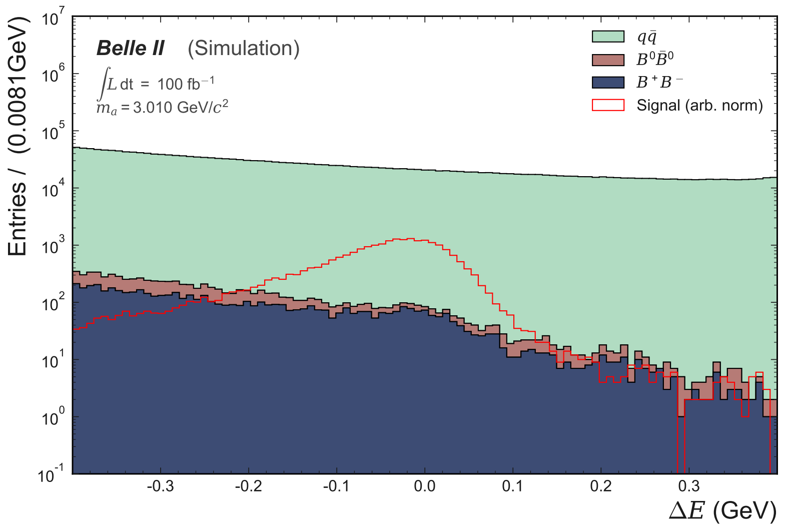
and overview of the mass space, this thesis displays the plots only for three masses $m_a \in \{1.510\,\mathrm{GeV}\,c^{-2}, 3.010\,\mathrm{GeV}\,c^{-2}, 4.510\,\mathrm{GeV}\,c^{-2}\}$. Figs. 5.1 to 5.3 displays the input distribution for the medium one of those masses, the other two are displayed in Appendix A.1.

- The M_{bc} distribution for the B events shows peaks at the rest mass of the B mesons. Continuum events show no peak as the $e^+e^- \to q\bar{q}$ decay involves no B meson.
- ΔE shows a flat distribution for continuum events. While the B events feature a peak at zero. For higher ALP masses, the ΔE distributions show prominent left tails due to shower leakage in the ECL for the higher energetic photons [1].
- The energy of the lower energetic photon $E_{\gamma'}$ peaks at zero for the background events, whereas the signal distribution shows a wide peak at a value rising with the ALP mass. As this peak overlaps with the background peak at zero for low masses, the separation power of this variable increases for higher masses [1].
- The $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$ is isotropic for B events due to the low thrust. In contrast, it clearly peaks at one for the continuum events due to their high boost.
- R_2 shows a Gaussian shape for the continuum samples due to their jet-like structure, while it is shifted towards zero for the B events.

In conclusion, the variables show great differences between continuum and B events. The separation between B background and signal events is only present in the photon energy $E_{\gamma'}$.

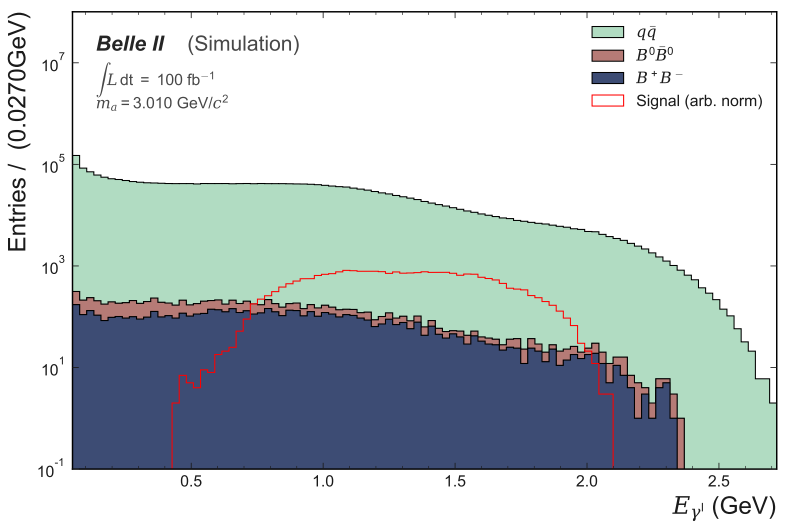


(a) $M_{\rm bc}$

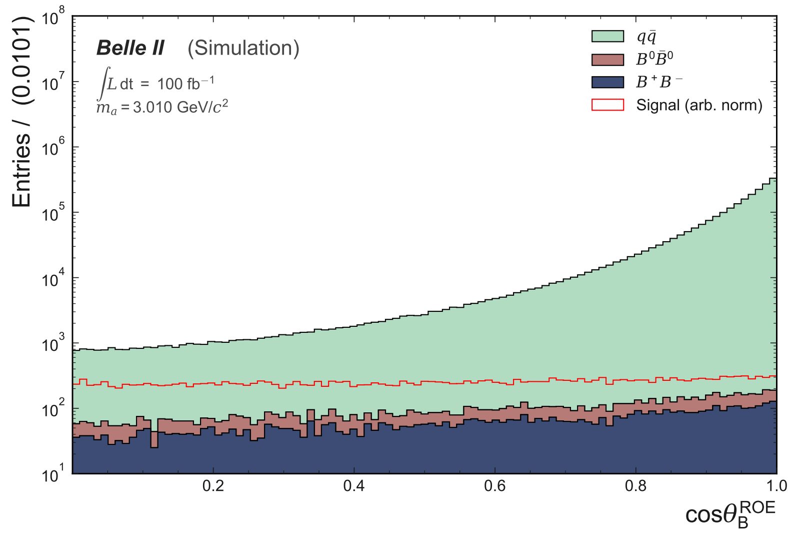


(b) ΔE

Figure 5.1.: These plots display the distributions of the input features $M_{\rm bc}$ and ΔE used in the main branch of the analysis. The histograms display the variables split by background types for the ALP mass $m_a = 3.010\,\text{GeV}\,c^{-2}$. The filled background curves for an integrated luminosity of $100\,\text{fb}^{-1}$ per type are stacked while the signal is unstacked in an arbitrary norm. Similar plots for the additional masses of $1.510\,\text{GeV}\,c^{-2}$ and $4.510\,\text{GeV}\,c^{-2}$ are displayed in Appendix A.1.

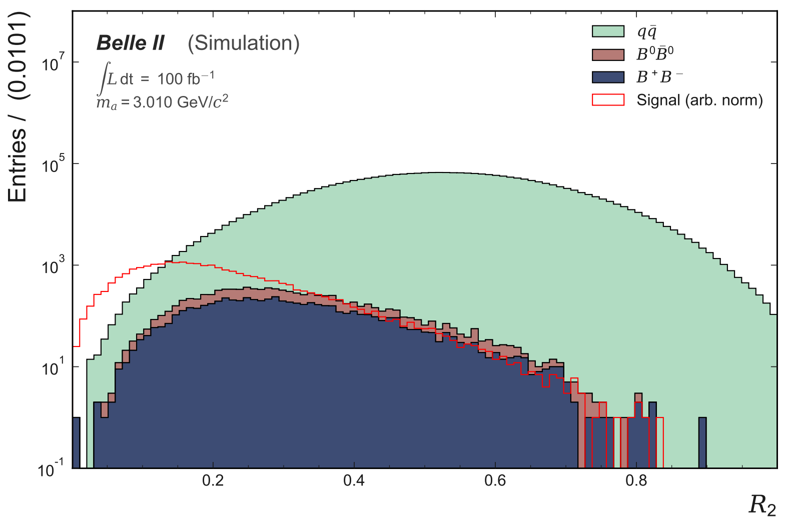


(a) E_{γ^l}



(b) $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$

Figure 5.2.: These plots display the distributions of the input features E_{γ^l} and $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$ used in the main branch of the analysis. The histograms display the variables split by background types for the ALP mass $m_a = 3.010\,\mathrm{GeV}\,c^{-2}$. The filled background curves for an integrated luminosity of $100\,\mathrm{fb}^{-1}$ per type are stacked while the signal is unstacked in an arbitrary norm. Similar plots for the additional masses of $1.510\,\mathrm{GeV}\,c^{-2}$ and $4.510\,\mathrm{GeV}\,c^{-2}$ are displayed in Appendix A.1.



(a) R_2

Figure 5.3.: This plot displays the distributions of the input feature R_2 used in the main branch of the analysis. The histograms display the variable split by background types for the ALP mass $m_a = 3.010\,\text{GeV}\,c^{-2}$. The filled background curves for an integrated luminosity of $100\,\text{fb}^{-1}$ per type are stacked while the signal is unstacked in an arbitrary norm. Similar plots for the additional masses of $1.510\,\text{GeV}\,c^{-2}$ and $4.510\,\text{GeV}\,c^{-2}$ are displayed in Appendix A.1.

5.1.2. Punzi-net Training with the First Architecture

The first approach for a hyperparameter setup is to use the original architecture proposed in the Punzi-net paper. Table 5.2 describes this architecture. Batch sizes, learning rates, and epochs for the training and pretraining are selected manually to train well with the given architecture.

Selecting the hyperparameters, as shown, leads to a good convergence in the pretraining loss (Fig. 5.4a) towards a low value without any signs of overtraining. The final training on the minimal detectable cross-section loss in Fig. 5.4b also shows good convergence without overtraining.
Fig. 5.5a displays the output classifier histogram after the final training. The histogram shows separation between signal and background, although few background samples feature high output values. These background events will remain after the selection due to the high output values. Furthermore, many signal samples feature a low output value close to zero. These signal events with a wrong predicted output will be removed by selections resulting in a lower signal efficiency. A similar histogram is also generated for untrained masses (Fig. 5.5b), which shows similar behaviour as the trained masses. This result for the untrained masses is not only strong evidence for proper training without overfitting but also a validation of the good generalization proposed by the Z' search [2].

The output classifier histograms are suitable for visualizing the separation of the signal from the different background types. However, they can not visualize or even quantify the performance for the different masses. Therefore, another test motivated by the original Punzi-net paper [2] is performed. On the same data set, ten networks are independently trained with the same hyperparameters. The multiple repetitions help to see instabilities in training, which might occur due to an insufficient set of parameters. Over these ten passes, the achieved signal efficiency and remaining background events are calculated after a fixed selection at an output value of 0.8. The selection of the numerical value 0.8 is motivated by including the region where the number of signal counts is rising towards the peak at one and to also exclude as much background as possible. This results in one efficiency and background value for each pass and each mass. For visualization, Fig. 5.6 shows the results averaged over the passes with their Gaussian standard deviations as error bars.

For signal efficiency, the shape of the curve is expected as the signal efficiency is highest for the medium masses while it drops for the outer masses. The deviation of the results also increases for outer masses, while it is almost zero for the medium mass range. Furthermore, the interpolation power on untrained masses can be clearly seen. The efficiency values for the untrained masses align well with the trained ones. The remaining background events after the selection also show a rise toward the medium masses. In comparison to the signal efficiency curve, the background events curve fluctuates more. The interpolation for untrained masses shows a considerably high deviation. Nevertheless, the intervals are still inside the value range for the trained masses.

This setup achieves a maximal signal efficiency of $19.5 \pm 0.4\,\%$ at an ALP mass of $3.510\,\mathrm{GeV}\,c^{-2}$ with 19 ± 1 remaining background events.

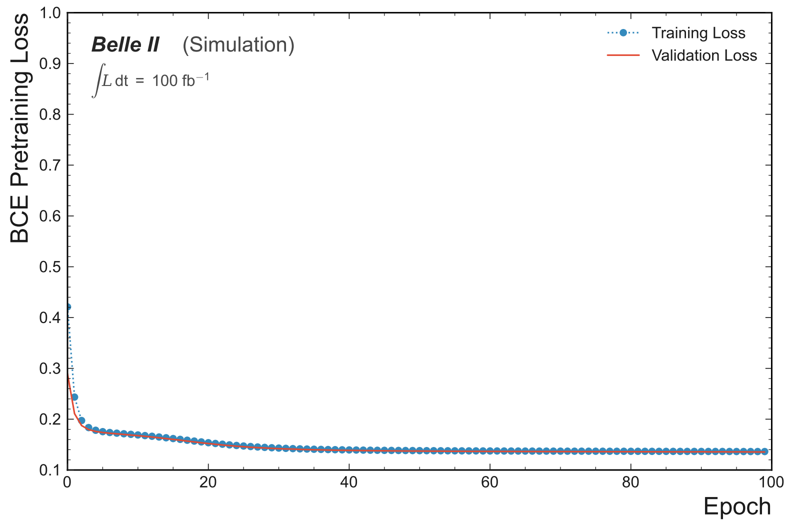
Table 5.2.: Overview of hyperparameters for the small architecture. The network architecture and activation functions are adopted from the Punzi-net publication that uses a similar architecture for four input features. The learning rate, epochs, and batch size for pretraining and final training are selected to train well with the given architecture.

(a) Architecture

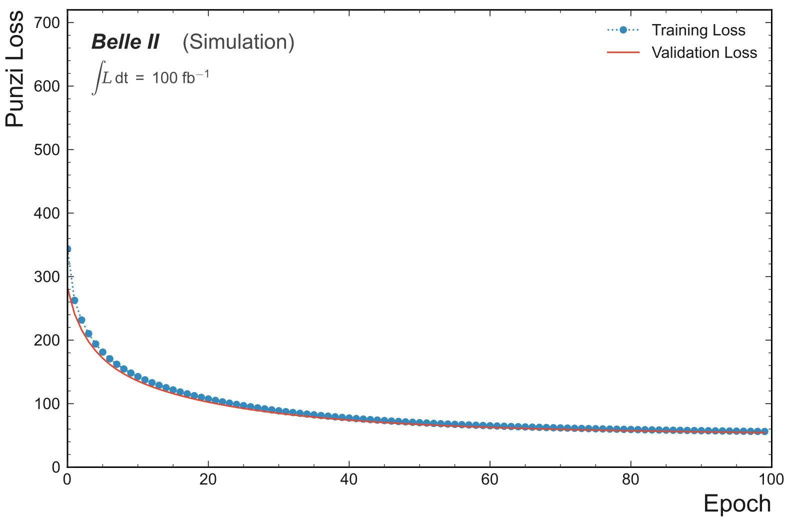
Layer	**Activation**	**Number of Nodes**
Input	tanh	5
1	tanh	10
2	tanh	5
3	sigmoid	1

(b) Training hyperparameters

Name	**Value**
Pretraining Batch Size	512
Pretraining Epochs	100
Pretraining Learning Rate	0.005
Training Batch Size	4096
Training Epochs	100
Training Learning Rate	0.0001

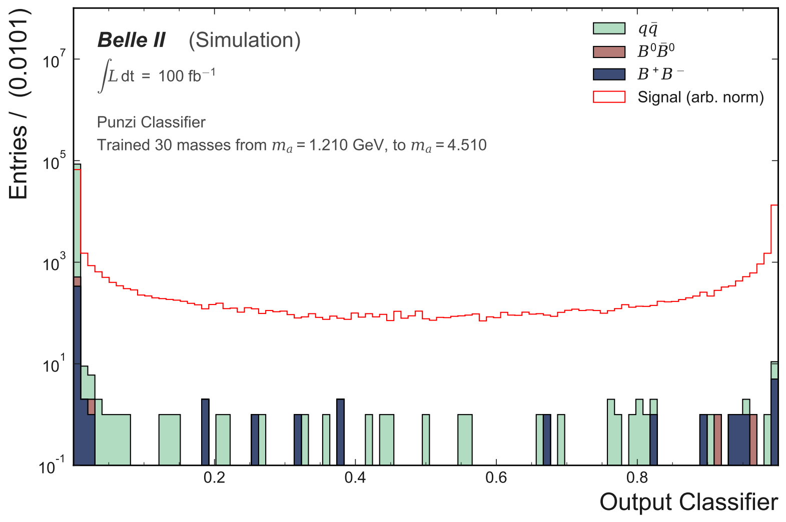


(a) BCE pretraining loss curves

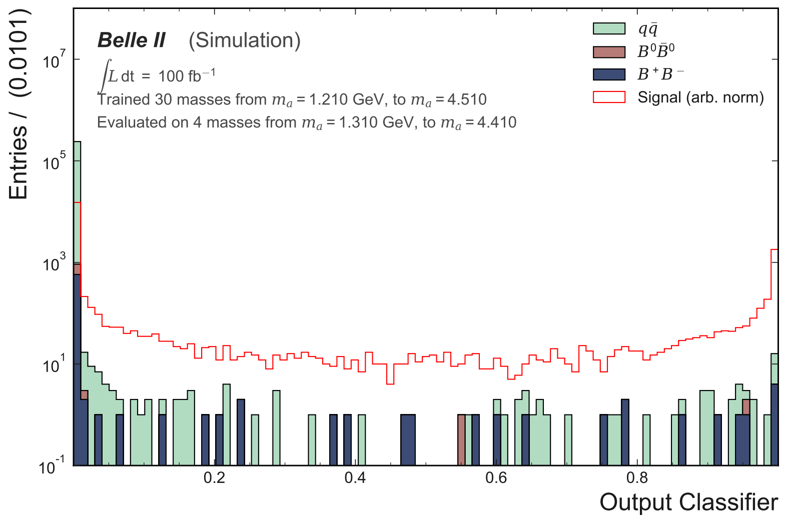


(b) Training Punzi-loss curves

Figure 5.4.: The training and validation loss curves for training Punzi-net on the initial set of variables with the small architecture. Both pretraining and training show a nominal convergence without obvious overtraining. The BCE-loss converges at a value of approximately 0.12 while the final training on the minimal detectable cross-section converges to ca. 60.

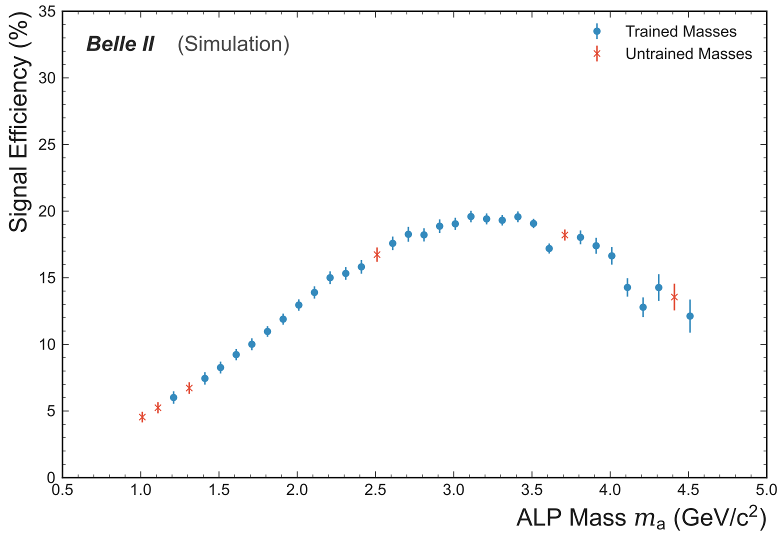


(a) Output for trained masses

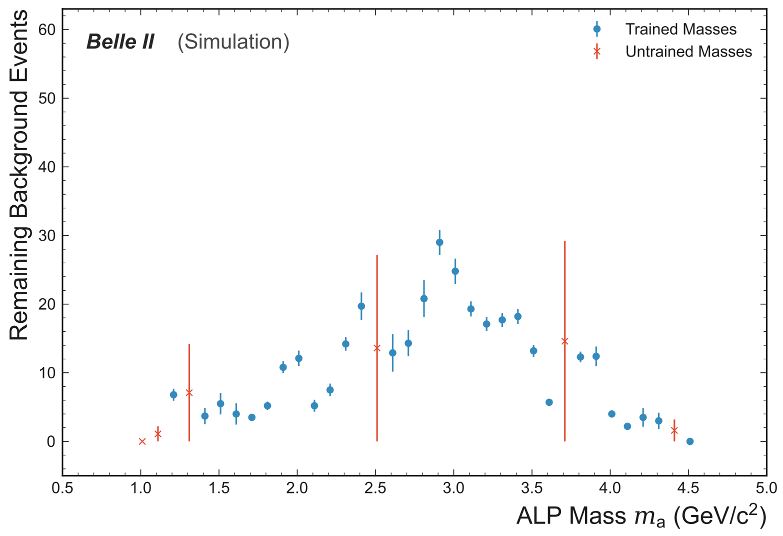


(b) Output for untrained masses

Figure 5.5.: The Plots show the output classifier distribution of Punzi-net for the trained masses (a) and untrained masses (b) on a logarithmic scale. The trained masses show a good background rejection with few background events with high output values close to one. The untrained masses show seemingly more background with high output values. The signal classification shows a high number of signals correctly classified at output values of approximately one, but also has some signals misidentified with medium or low output values.



(a) Signal efficiency



(b) Background events

Figure 5.6.: The plots show the performance of the small Punzi-net architecture on the original variable set. To obtain the signal efficiency (a) and number of background events (b), the network trains ten times and the results after selecting output values greater than 0.8 are averaged over the ten different trainings. The signal efficiency displays a smooth curve rising from 5 % to a maximum value of about 20 % for the medium masses. The untrained mass hypotheses align well with the curve of the trained masses. The remaining background events after the selection displays a similar shape, rising from 0 towards 30 for the medium masses and decreasing again for the high masses. Although in comparison to the signal curve, it varies more. The untrained masses feature a high standard deviation for the number of background events.

5.1.3. Training with an Improved Architecture

Different architectures are tested to check if the results of Punzi-net are improvable. During the studies, the trend was that deeper and wider networks tend to perform better. This can be explained by their bigger generalization qualities [19]. Out of the tested architectures, the one displayed in Table 5.3 performs the best on the original set of variables. This setup trains sufficiently with the same set of hyperparameters as the smaller architecture according to the loss curves (Fig. 5.7). Compared to the previous setup, this one shows two main differences in the loss curves:

1. **Lower loss values:** In comparison, the bigger architecture converges to lower loss values, representing fewer errors in the output classification.
2. **Validation loss below training loss:** The validation loss curve is well below the training loss in the first 30 epochs of the final training. This might be due to statistical fluctuations in this particular training or come from the procedure of loss calculation. As the validation loss is only calculated at the end of each epoch, the network has trained the whole dataset before the validation loss is calculated. At the same time, the training loss is averaged over all batches in the epoch. Therefore, the training loss picks up the changes in the network during the whole epoch. In contrast, the validation loss only uses the last and commonly best iteration of the NN.

The output classifiers in Fig. 5.8 strengthen the assumption that the new setup can outperform the previous one. The distributions for trained and untrained masses show fewer background events in the medium output value regions. In addition, more signal is classified with an output value near 1.
The result of the ten training test in Fig. 5.9 is also encouraging, as the resulting maximum signal efficiency is $24.6 \pm 0.2\,\%$ at a background level of 13 ± 3 for a mass of $3.510\,\mathrm{GeV}\,c^{-2}$, which is an improvement over the result from the first architecture (Section 5.1.2).

These results show that better architectures can improve performance. Furthermore, these outcomes are probably not the best ones possible, as the parameter optimization was done manually and not every parameter could be tested.

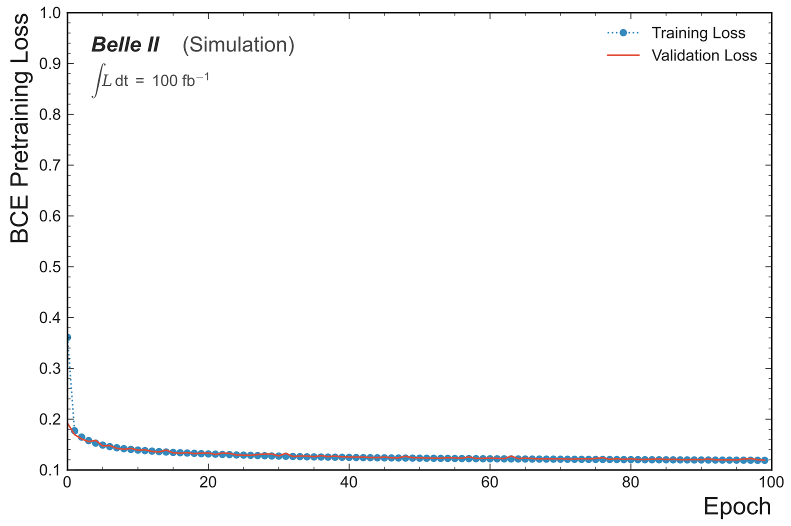
Table 5.3.: Overview of hyperparameters for the enlarged architecture. According to the approximation theorem, this should improve the performance. [19] To obtain this architecture, tests with different architectures and hyperparameters are performed manually, and the best-found configuration is selected.

(a) Architecture

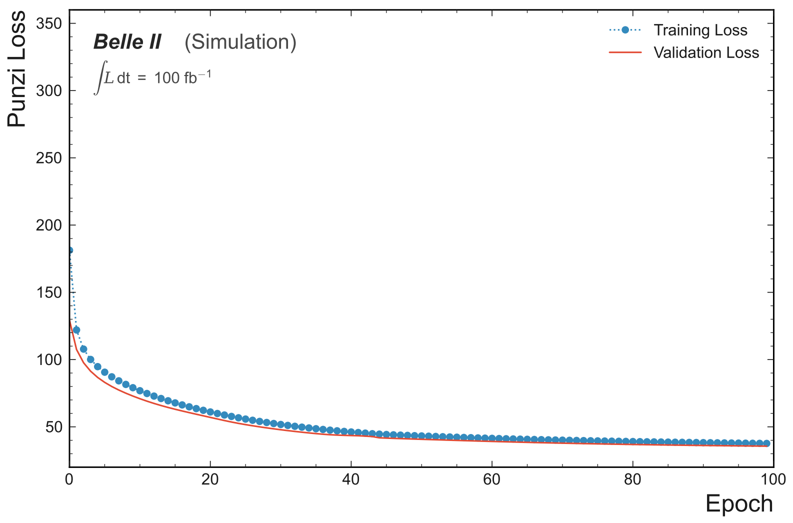
Layer	**Activation**	**Number of Nodes**
Input	tanh	5
1	tanh	16
2	tanh	32
3	tanh	32
4	tanh	8
5	tanh	4
6	sigmoid	1

(b) Training hyperparameters

Name	**Value**
Pretraining Batch Size	512
Pretraining Epochs	100
Pretraining Learning Rate	0.005
Training Batch Size	4096
Training Epochs	100
Training Learning Rate	0.0001

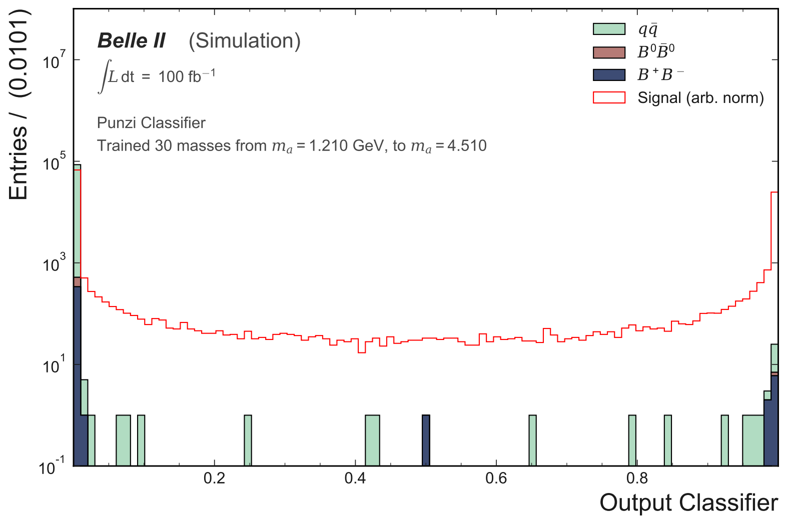


(a) BCE pretraining loss curves

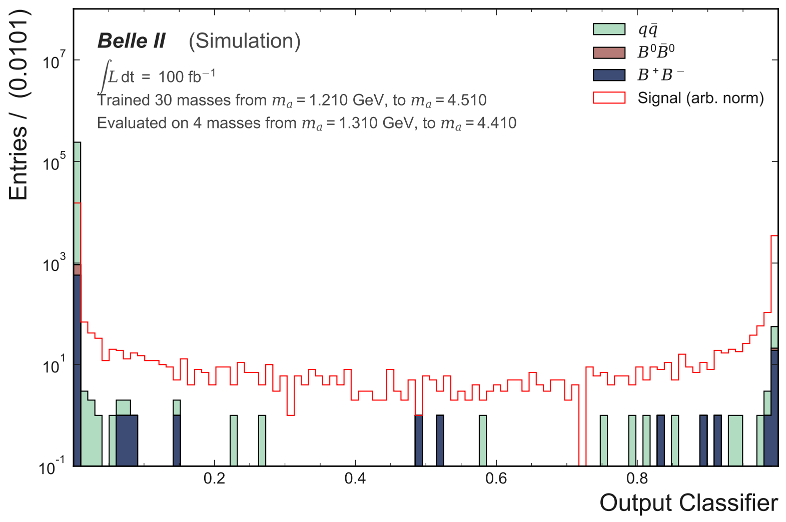


(b) Training Punzi-loss curves

Figure 5.7.: These plots show the training and validation loss curves for the training on the original set of variables with the enlarged architecture. The pretraining loss displays fast convergence to a value of 0.12 after about 40 epochs. Final training on the Punzi-loss also converges to a low value of approximately 40. Both trainings show no obvious signs of overtraining.

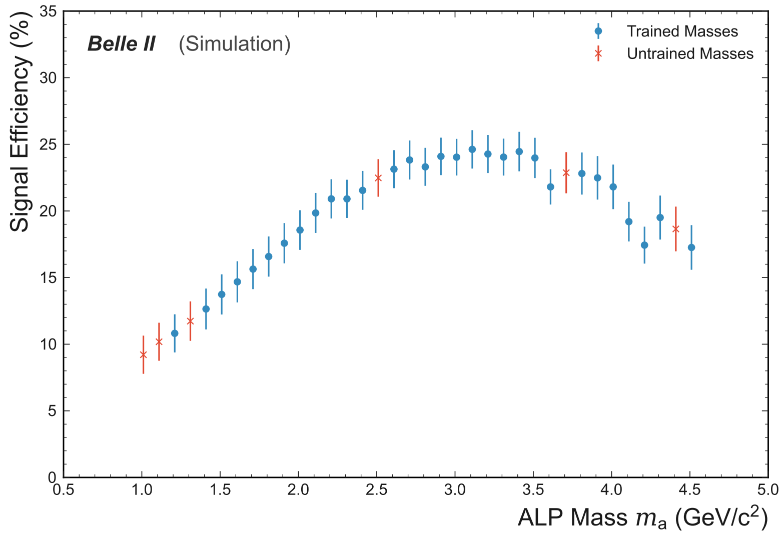


(a) Output for trained masses

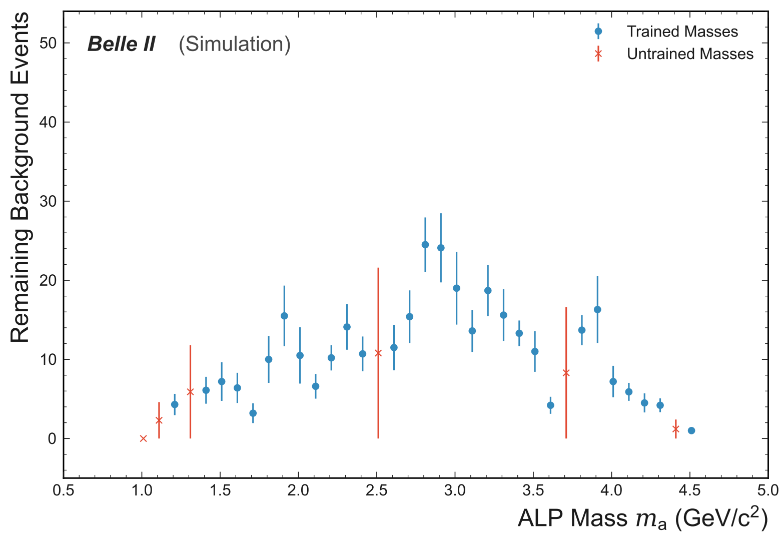


(b) Output for untrained masses

Figure 5.8.: The plots show the output distribution of Punzi-net in the improved setup for the trained masses (a) and untrained masses (b). Both show very few background events with high output classifier values. Furthermore, both plots display good signal classification with few signal events in the medium mass regions and peaks at output values of one and zero.



(a) Signal efficiency



(b) Background events

Figure 5.9.: The plot shows the performance of the enlarged Punzi-net architecture on the original variable set. To obtain the signal efficiency (a) and background events (b), the network trains ten times and the results after a fixed selection at an output greater 0.8 are averaged. The signal efficiency this setup achieves averaged over all masses is 19±5 % at a background level of 10±7. The untrained masses show large uncertainties in the background events plot compared to the trained ones.

5.2. Training on Additional Variables

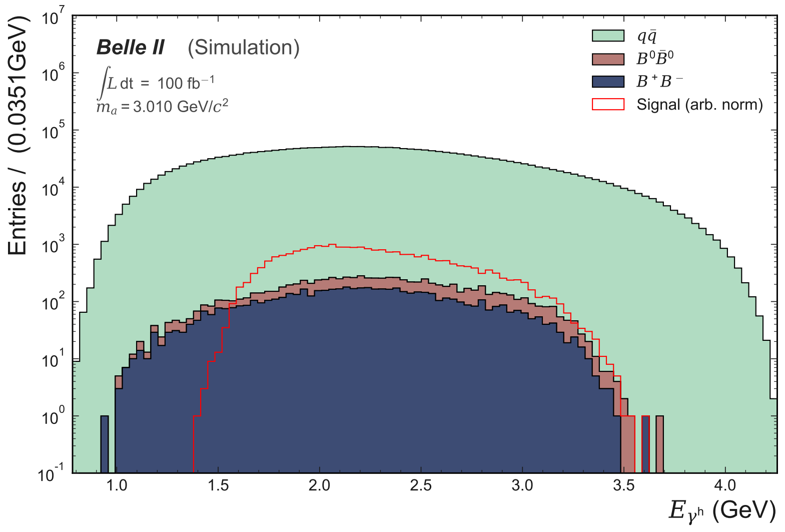
These features are motivated by leaving M_{bc} and ΔE out of the feature set to use them in the proceeding analysis. Table 5.4 shows the feature set used for training. This set is only one of the possible combinations of the additional variables from Chapter 2. Figs. 5.10 and 5.11 display the distributions of the new input features for the medium mass of $3.010\,\mathrm{GeV}\,c^{-2}$ while the distributions for the lower and higher masses are displayed in Appendix A.2. The distribution of the features R_2, $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$, and E_{γ^l} which are used previously are in Figs. 5.2 and 5.3 and Appendix A.1.

- The distributions of R_2, E_{γ^l}, and $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$ are already described in Section 5.1. Of them, only E_{γ^l} shows separation between B background and signal events.
- E_{γ^h} shows a broad peak for signal samples, similar to E_{γ^l}. For the background events, E_{γ^h} takes broader Gaussian-shaped distributions.
- The signal in E_{Kaon} shows a plateau at lower energies for higher ALP masses. This inverse connection can be explained by relativistic energy conservation, as the creation of the ALP requires most of the B mesons energy, leaving less kinetic energy for the kaon. The backgrounds again show broader distributions.
- The kaon's transverse momentum p_{Kaon}^{T} shows the same behaviour as E_{Kaon}.

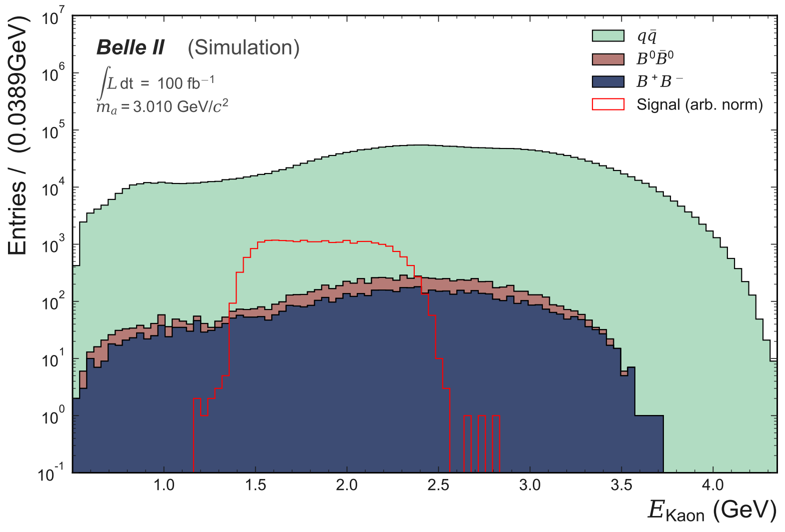
Besides the clear separation between continuum and B events, this feature set shows separation between B background and signal in more variables than the previous set.

Table 5.4.: This table shows the additional features set of 6 variables, with the purpose of leaving M_{bc} and ΔE out of the training variables.

Feature	**Variable Name**
1	R_2
2	E_{γ^l}
3	E_{γ^h}
4	E_{Kaon}
5	$\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$
6	p_{Kaon}^{T}

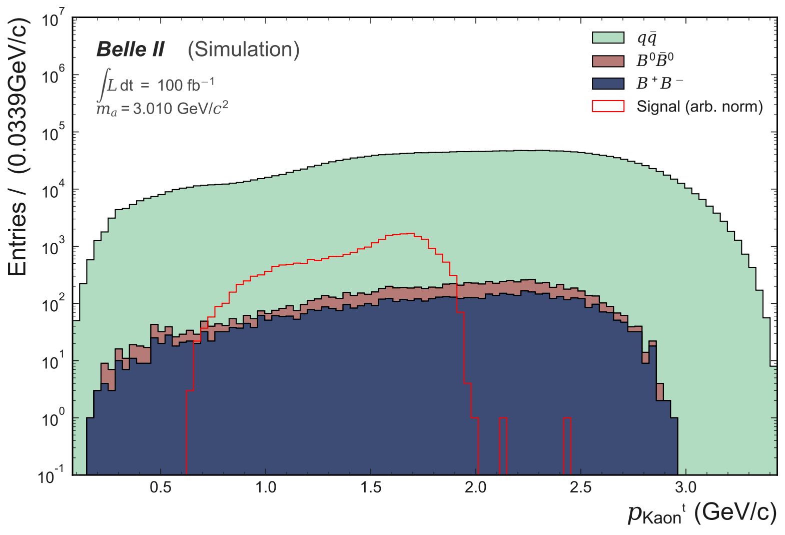


(a) E_{γ^h}



(b) E_{Kaon}

Figure 5.10.: These plots display the distributions of the additional input feature set tested. The histograms display the variables E_{γ^h} and E_{Kaon} split by types for the ALP mass $m_a = 3.010\,\text{GeV}\,c^{-2}$. The filled background curves are stacked while the arbitrary norm signal is unstacked. Equivalent plots for the additional masses of $1.510\,\text{GeV}\,c^{-2}$ and $4.510\,\text{GeV}\,c^{-2}$ are displayed in Appendix A.2.



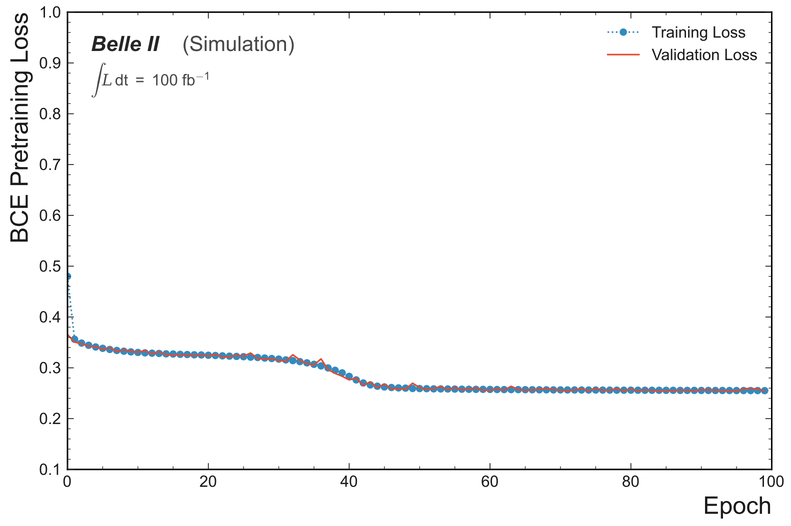
(a) p^T_{Kaon}

Figure 5.11.: This plot displays the distributions of the additional input feature set tested. The histograms display the variable p^T_{Kaon} split by types for the ALP mass $m_a = 3.010\,\text{GeV}\,c^{-2}$. The filled background curves are stacked while the arbitrary norm signal is unstacked. Equivalent plots for the additional masses of $1.510\,\text{GeV}\,c^{-2}$ and $4.510\,\text{GeV}\,c^{-2}$ are displayed in Appendix A.2.

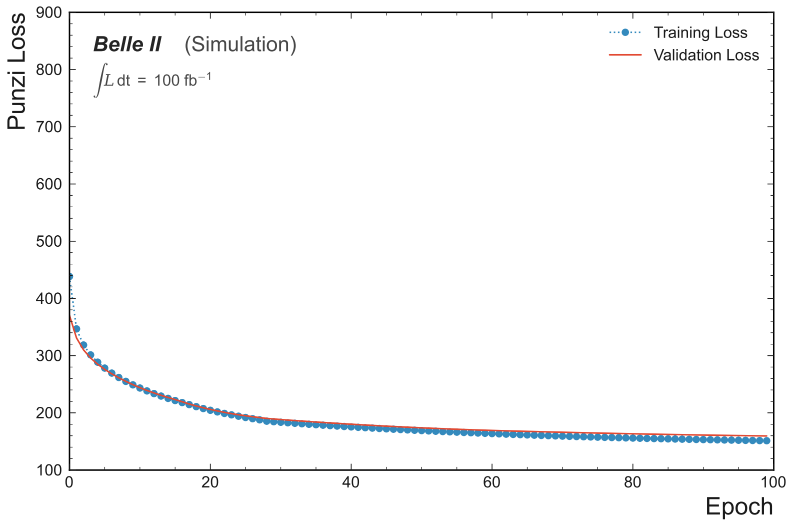
5.2.1. Training on the New Feature Set

Despite the fact that the new variables bring additional information and apparent differences in the distributions, Punzi-net, in the current state, seems not to learn a sufficient representation of them. During the training of the enlarged architecture from Table 5.3 many deteriorations were seen.
In pretraining, the network has a high error rate which manifests in high loss values through the training. Compared to the previous variable set which converged at a BCE loss of approximately 0.12 (Fig. 5.7a), the new variables lead to a convergence at approximately 0.25 shown in Fig. 5.12a.
Furthermore, challenges in learning influence the neural networks output histograms in Fig. 5.13. These already show a high level of misclassified background with output values greater than 0.8, while the signal loss is also more extensive than before. Training the neural network ten times and averaging the results for signal efficiency and background events (Fig. 5.14) quantifies these deductions. The background curve shows the high number of remaining background events with up to 75 events at $m_a = 2.8\,\mathrm{GeV}\,c^{-2}$.

Fig. 5.14a shows that the resulting signal efficiency is also lower compared to the case where only the initial variables were used in the training. The training achieves a maximum signal efficiency $13.3 \pm 0.8\,\%$ at a mass of $3.210\,\mathrm{GeV}\,c^{-2}$ and a background level of 76 ± 9 events. An additional indication of insufficient training is the generalization to untrained masses. Notably, the signal efficiency shows this behaviour, with the untrained masses falling short behind the trained ones. Further tests on this feature set can provide additional insight as to whether or not the challenges observed when training this feature set are native to the features themselves or if they can be solved by further optimization.

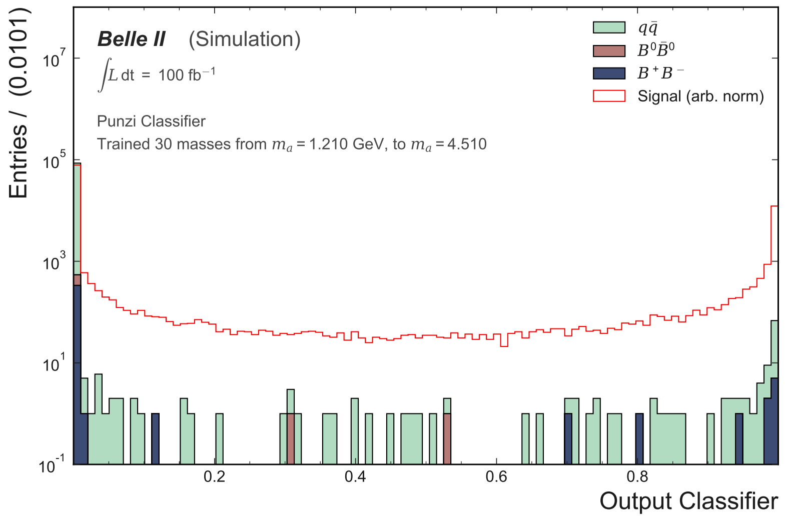


(a) BCE pretraining loss curves

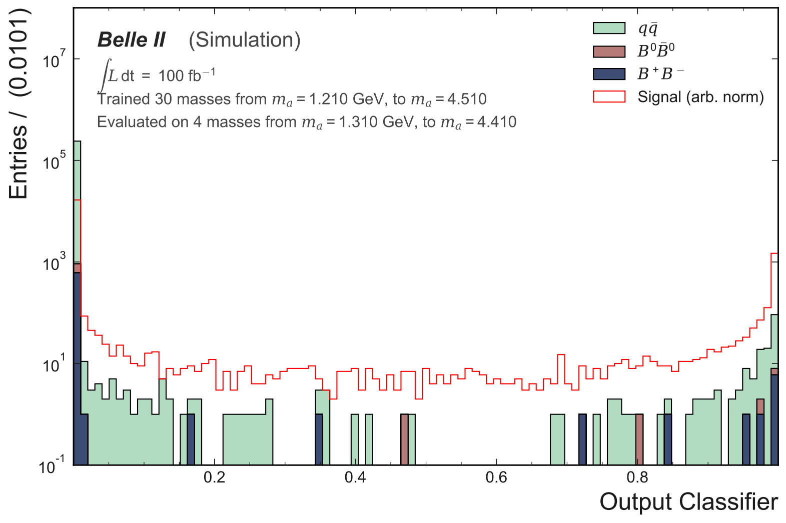


(b) Training Punzi-loss curves

Figure 5.12.: These plots show the training and validation loss curves for the training on the additional set of variables with the enlarged architecture. The pretraining on the BCE-loss function converges to around 0.26 with a step between epoch 30 and 45. The training on the Punzi-loss function displays smooth exponential shape conversion to a value of 150. During the final 40 epochs, the validation loss rises slightly over the training loss, which could be a sign of overtraining.

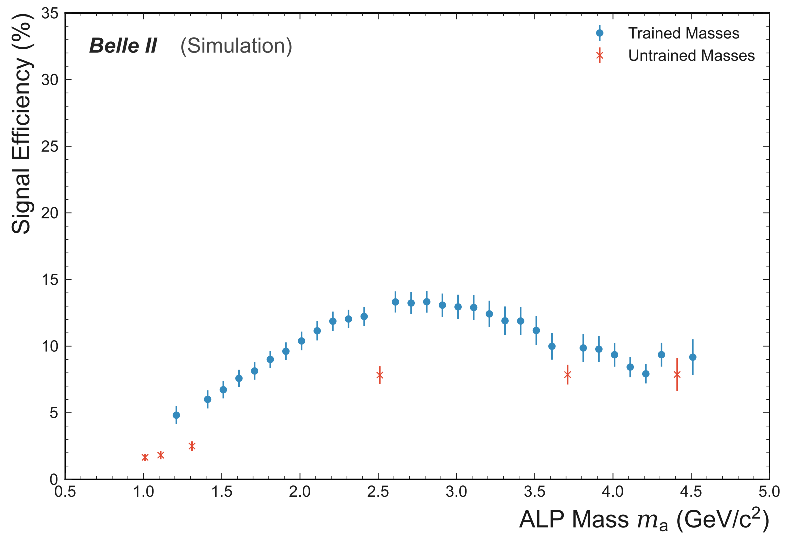


(a) Output for trained masses

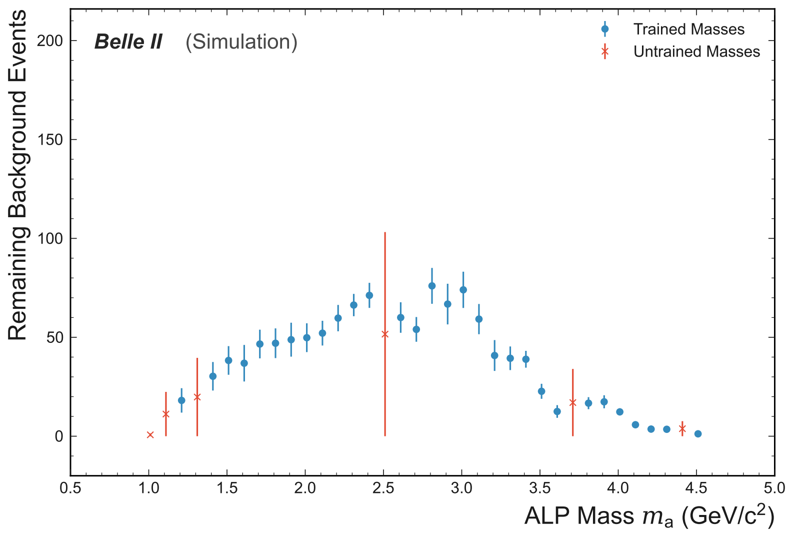


(b) Output for untrained masses

Figure 5.13.: The plots show the output distribution of Punzi-net in the improved setup for the trained masses (a) and untrained masses (b) with the additional set of parameters. The background histograms feature a high number of background events with medium and high output classifier values leading to lower signal purities. The signal distributions display a high number of signal counts with low output values.



(a) Signal efficiency



(b) Background events

Figure 5.14.: The plots display the performance of the enlarged Punzi-net architecture on the additional variable set. The network trains ten times to obtain the signal efficiency (a) and background events (b) and the results after a fixed selection at 0.8 are averaged. The signal efficiency this setup achieves averaged over all masses is $9 \pm 3\,\%$ at a background level of 35 ± 25. The background events remaining after the selection again display a maximum for the medium masses at approximately 80 events and a fluctuating reduction for smaller and larger masses down to 0 events. Untrained masses feature higher standard deviations in the background events, compared to the trained ones. The achieved signal efficiency curve shows the known shape for the trained masses rising from about $5\,\%$ to $15\,\%$ for the medium masses and falling down to around $10\,\%$ for high masses. Untrained masses have a signal efficiency that does not align with the ones for trained masses.

6. Conclusion and Outlook

This thesis presents an improved method for candidate selection in the search for the new physics signature of the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ decay at Belle II. To implement Punzi-net into the $B^{\pm} \to K^{\pm}a,\ a \to \gamma\gamma$ analysis, this thesis applies changes to the data preparation (mainly through data loading) and candidate selection. Furthermore, this thesis implements validation loss for pretraining and training to monitor performance and check for overtraining. The modifications contain a batched training for the final training on the minimal detectable cross-section. Additionally, this thesis adds further functions for visualization and quantification of the neural network's classification performance.

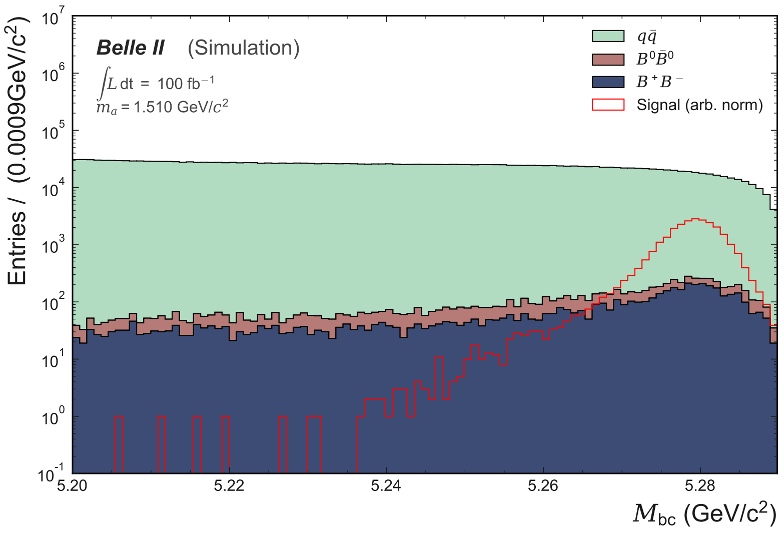
In this state, the network achieves signal efficiencies of 10 % to 25 % with a background reduction to less than 30 events for an integrated luminosity of $100\,\mathrm{fb}^{-1}$. Furthermore, tests with untrained masses show a good interpolation capability for Punzi-net, strengthening the hypothesis from the previous use in the $e^{+}e^{-} \to \mu^{+}\mu^{-}Z'$ search that Punzi-net can interpolate well to untrained masses [2].

Additionally, this thesis performs basic hyperparameter studies. A possible improvement in the hyperparameter handling could be the implementation of an automated optimization procedure, e. g. a grid search in the hyperparameter space. A more streamlined implementation of Punzi-net, which is easy to use and runnable on a computing grid, would significantly improve the capabilities for automated optimization and future analyses.
The extended hyperparameter studies could also probe the hyperparameters for the additional feature set, which is challenging for the current implementation.
Additionally, the Punzi-net selection could be established together with a previous or subsequent selection by another classifier. For example, a first network is trained for continuum suppression and a second dedicated network for B background suppression.

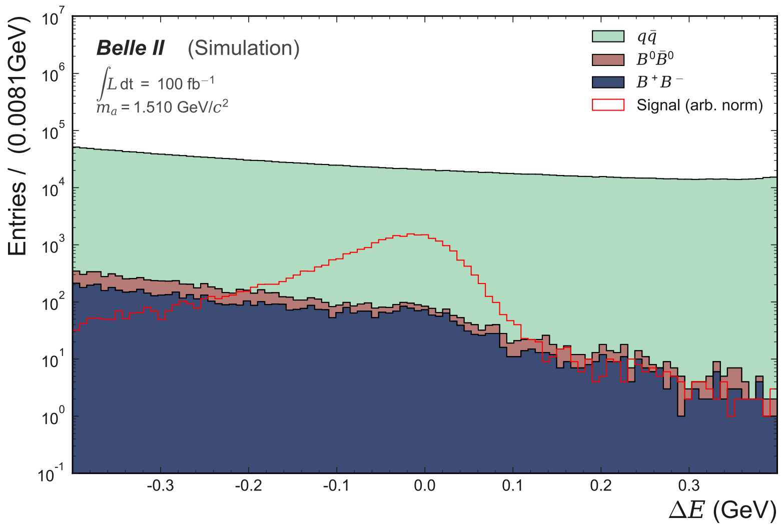
This thesis examines promptly decaying axion-like particles above the η' resonance. Further studies should also look at non-zero lifetimes and lower masses to probe if the novel approach is also applicable in these regions.

A. Appendix: Input Variable Distributions

A.1. Main Feature Set

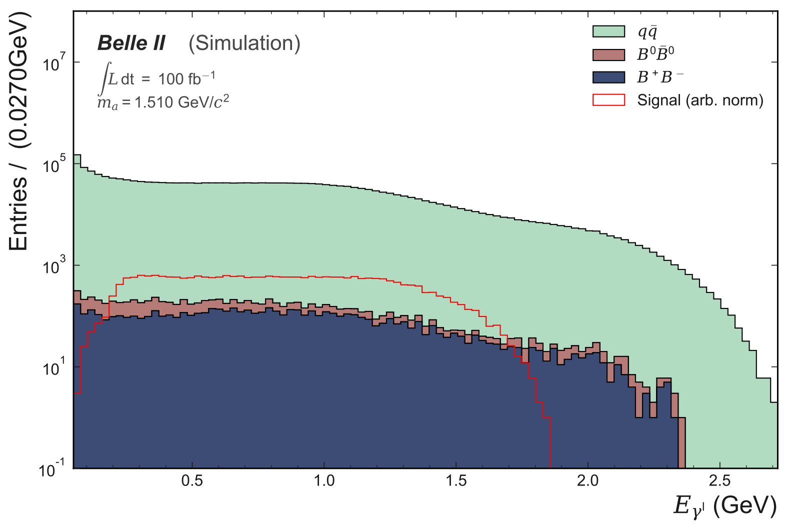


(a) $M_{\rm bc}$

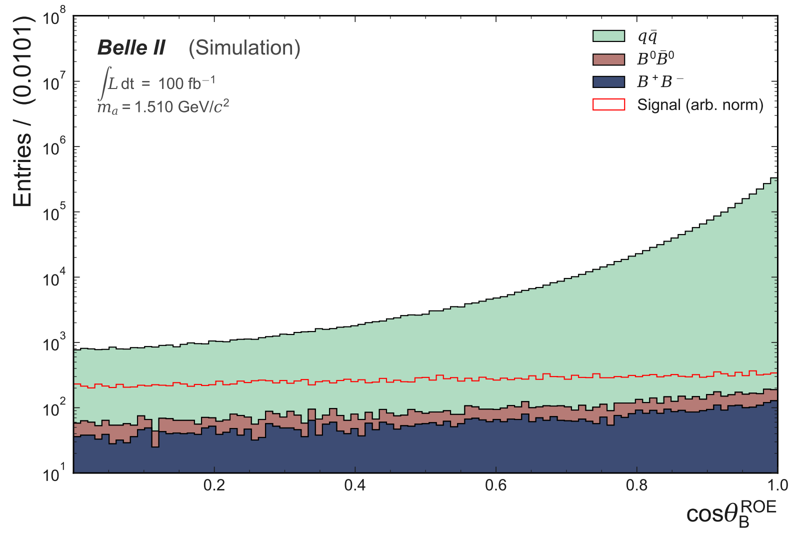


(b) ΔE

Figure A.1.: These plots display the distributions of the input features used in the main branch of the analysis. The histograms display the variables $M_{\rm bc}$ and ΔE split by types for the ALP mass $m_a = 1.510\,\text{GeV}\,c^{-2}$. The filled background curves are shown stacked while the signal in arbitrary norm is unstacked.

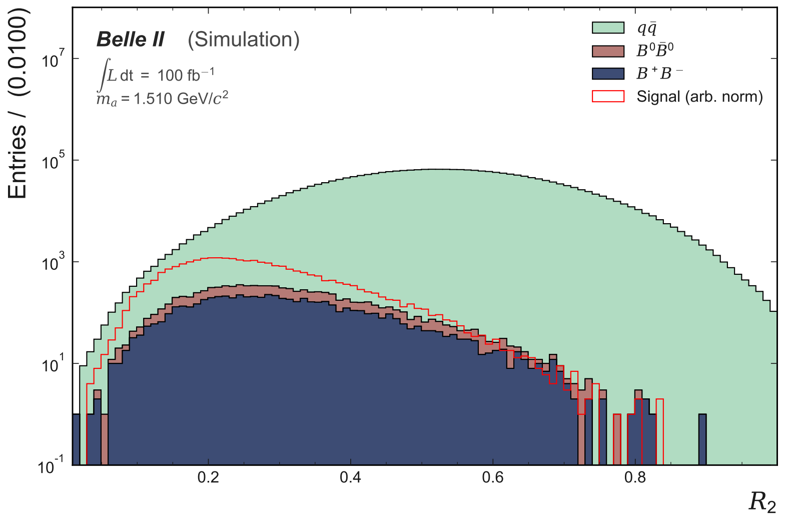


(a) E_{γ^l}



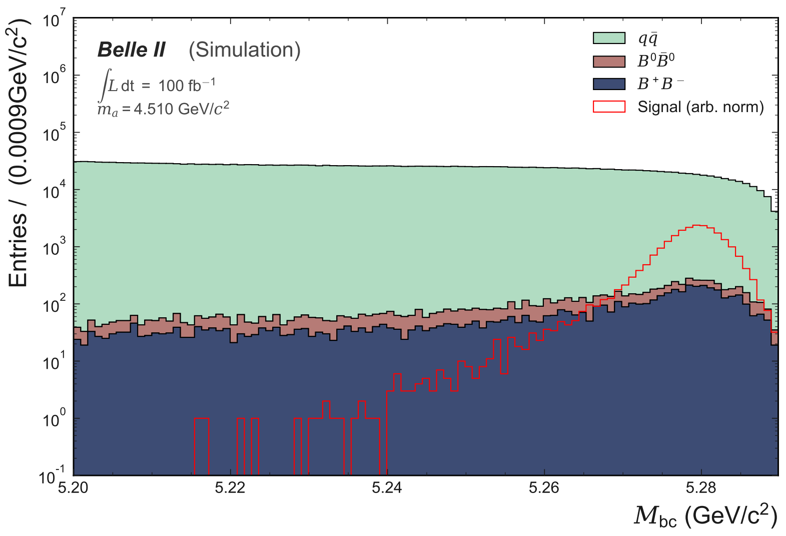
(b) $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$

Figure A.2.: These plots display the distributions of the input features used in the main branch of the analysis. The histograms display the variables E_{γ^l} and $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$ split by types for the ALP mass $m_a = 1.510\,\mathrm{GeV}\,c^{-2}$. The filled background curves are shown stacked while the signal in arbitrary norm is unstacked.

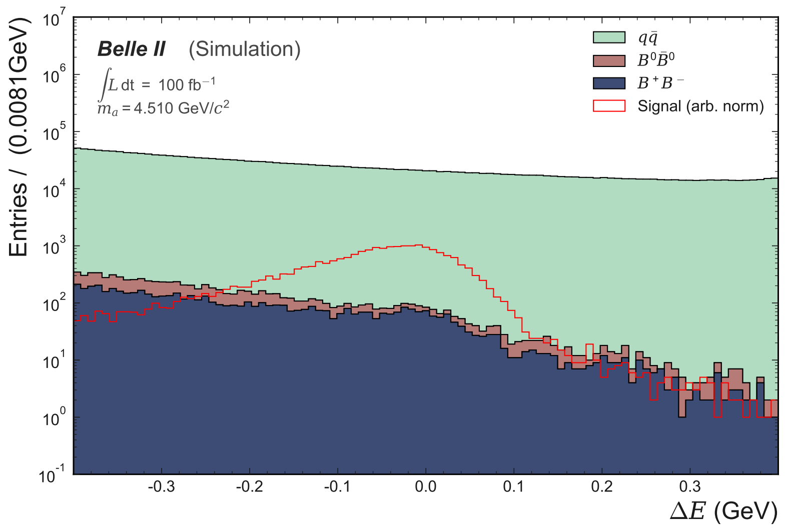


(a) R_2

Figure A.3.: This plot displays the distributions of the input features used in the main branch of the analysis. The histograms display the variable R_2 split by types for the ALP mass $m_a = 1.510\,\mathrm{GeV}\,c^{-2}$. The filled background curves are shown stacked while the signal in arbitrary norm is unstacked.

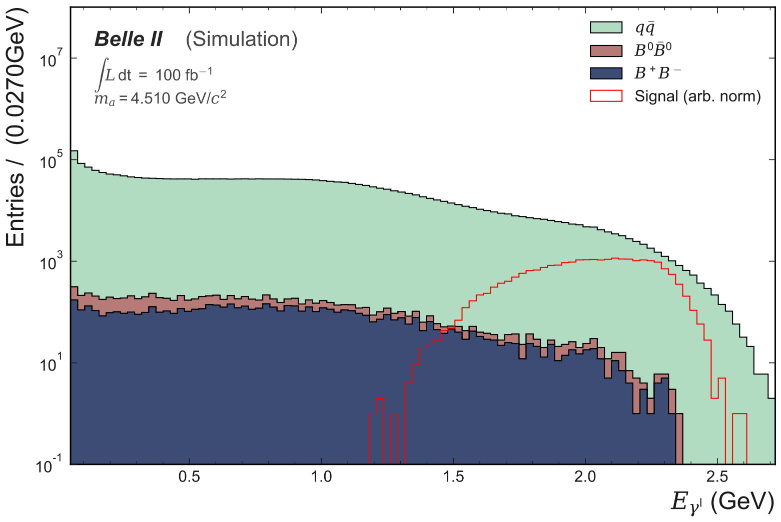


(a) $M_{\rm bc}$

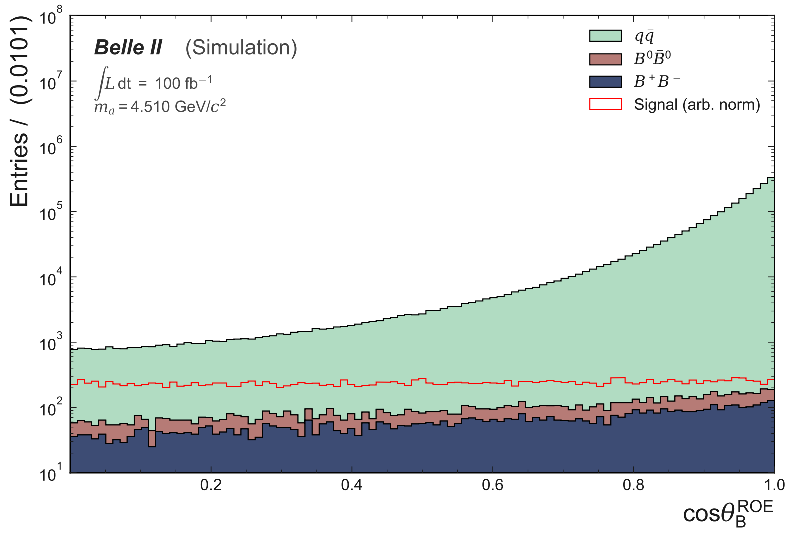


(b) ΔE

Figure A.4.: These plots display the distributions of the input features used in the main branch of the analysis. The histograms display the variables $M_{\rm bc}$ and ΔE split by types for the ALP mass $m_a = 4.510\,\mathrm{GeV}\,c^{-2}$. The filled background curves are shown stacked while the signal in arbitrary norm is unstacked.

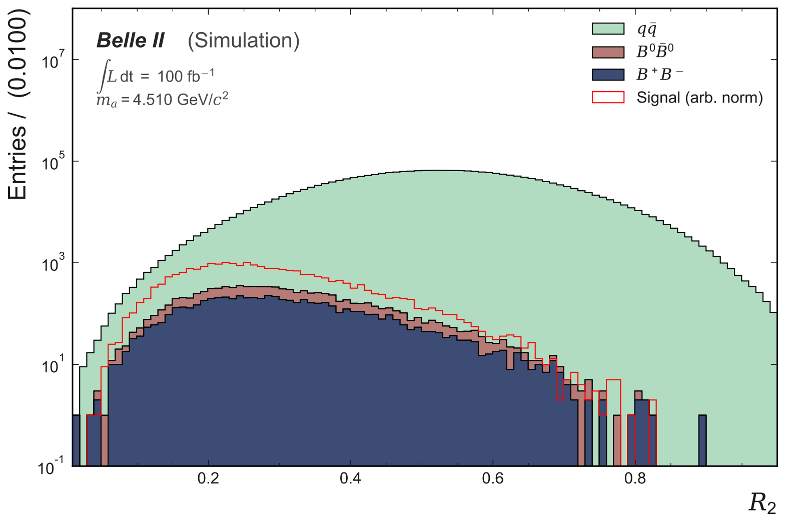


(a) E_{γ^l}



(b) $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$

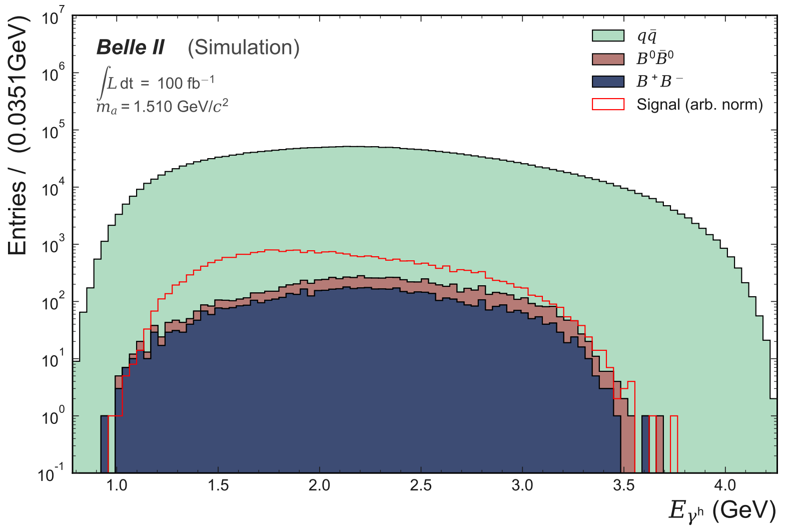
Figure A.5.: These plots display the distributions of the input features used in the main branch of the analysis. The histograms display the variables E_{γ^l} and $\cos\theta_{\mathrm{B}}^{\mathrm{ROE}}$ split by types for the ALP mass $m_a = 4.510\,\mathrm{GeV}\,c^{-2}$. The filled background curves are shown stacked while the signal in arbitrary norm is unstacked.



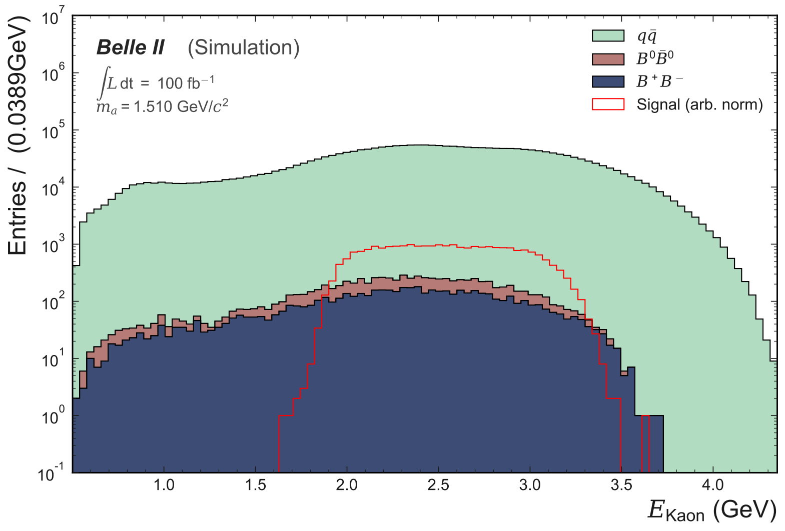
(a) R_2

Figure A.6.: This plot displays the distributions of the input features used in the main branch of the analysis. The histograms display the variable R_2 split by types for the ALP mass $m_a = 4.510\,\mathrm{GeV}\,c^{-2}$. The filled background curves are shown stacked while the signal in arbitrary norm is unstacked.

A.2. Additional Feature Set

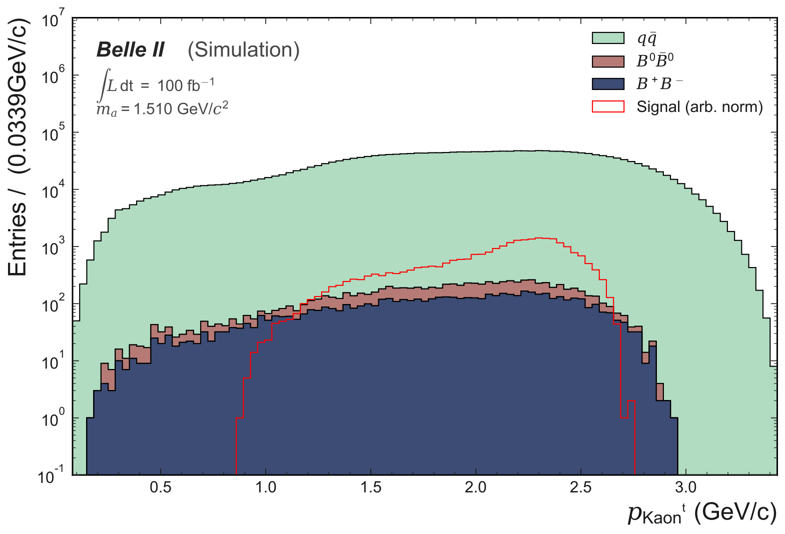


(a) E_{γ^h}



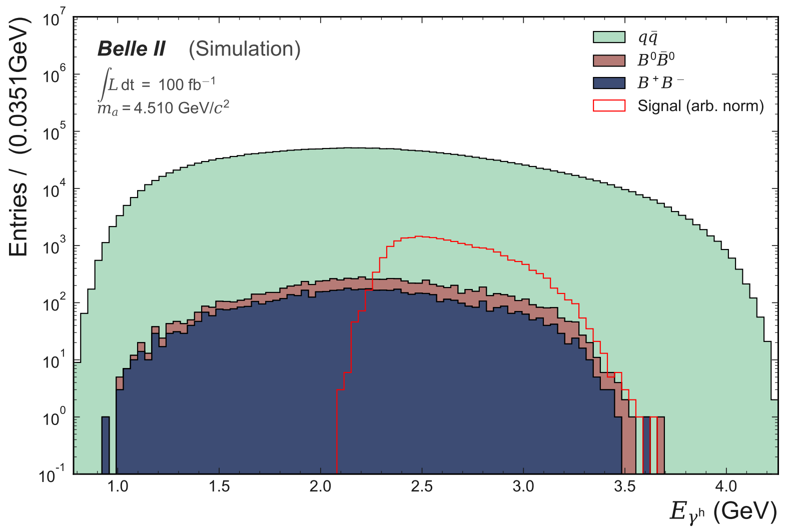
(b) E_{Kaon}

Figure A.7.: These plots display the distributions of the additional input feature set tested. The histograms display the variables E_{γ^h} and E_{Kaon} split by types for the ALP mass $m_a = 1.510\,\text{GeV}\,c^{-2}$. The filled background curves are stacked while the arbitrary norm signal is unstacked.

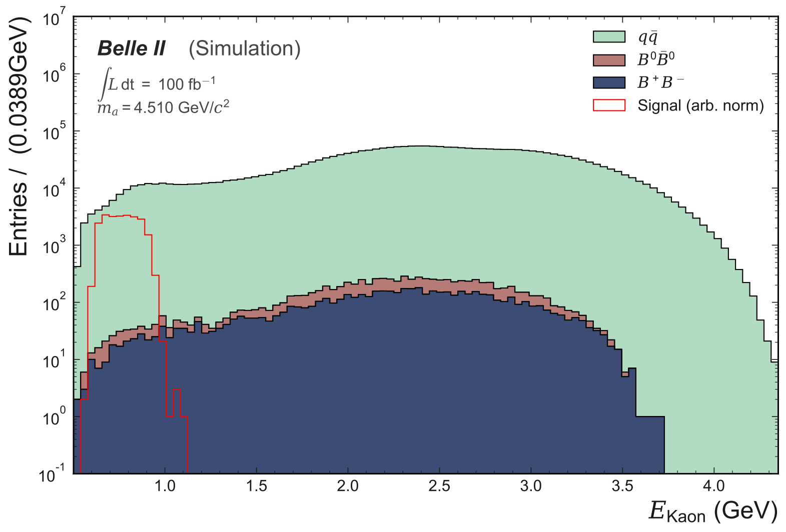


(a) p^T_{Kaon}

Figure A.8.: These plots display the distributions of the additional input feature set tested. The histogram displays the variable p^T_{Kaon} split by types for the ALP mass $m_a = 1.510\,\text{GeV}\,c^{-2}$. The filled background curves are stacked while the arbitrary norm signal is unstacked.

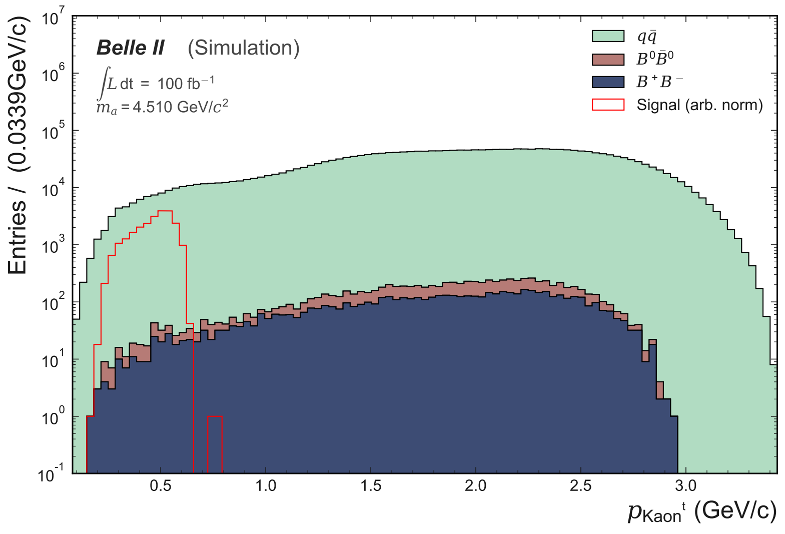


(a) E_{γ^h}



(b) E_{Kaon}

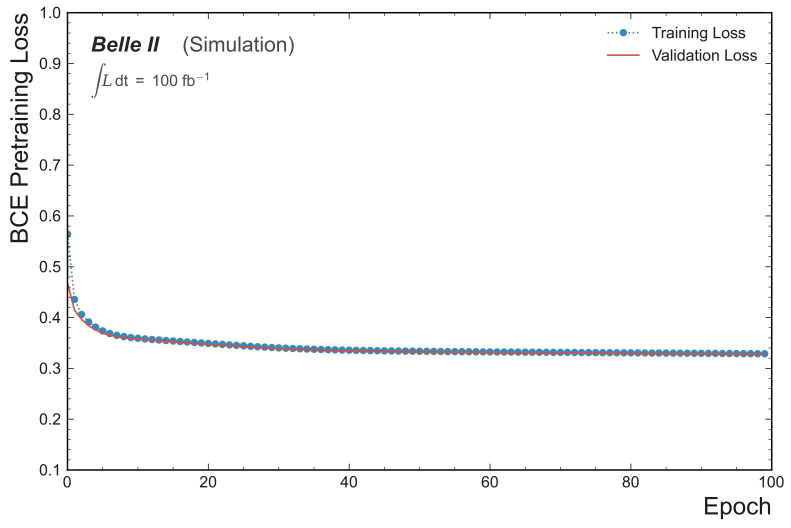
Figure A.9.: These plots display the distributions of the additional input feature set tested. The histograms display the variables E_{γ^h} and E_{Kaon} split by types for the ALP mass $m_a = 4.510\,\text{GeV}\,c^{-2}$. The filled background curves are stacked while the arbitrary norm signal is unstacked.



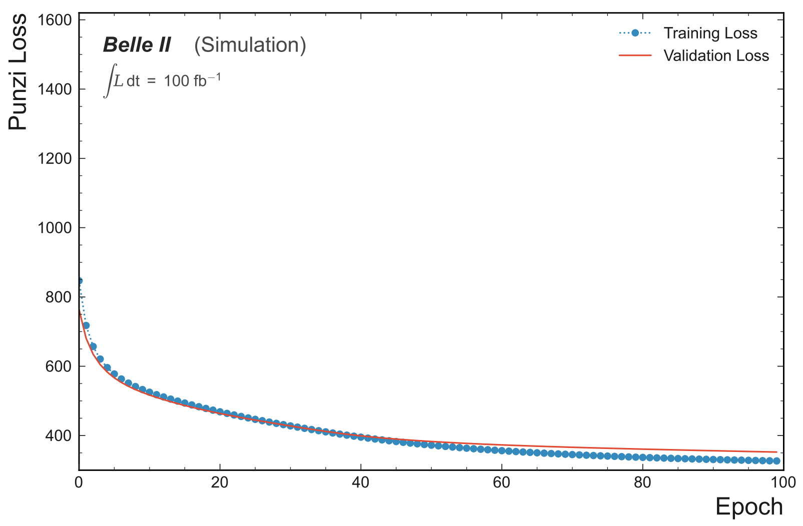
(a) p^{T}_{Kaon}

Figure A.10.: These plots display the distributions of the additional input feature set tested. The histogram displays the variable p^{T}_{Kaon} split by types for the ALP mass $m_a = 4.510\,\mathrm{GeV}\,c^{-2}$. The filled background curves are stacked while the arbitrary norm signal is unstacked.

B. Training the Additional Variables on the Small Architecture

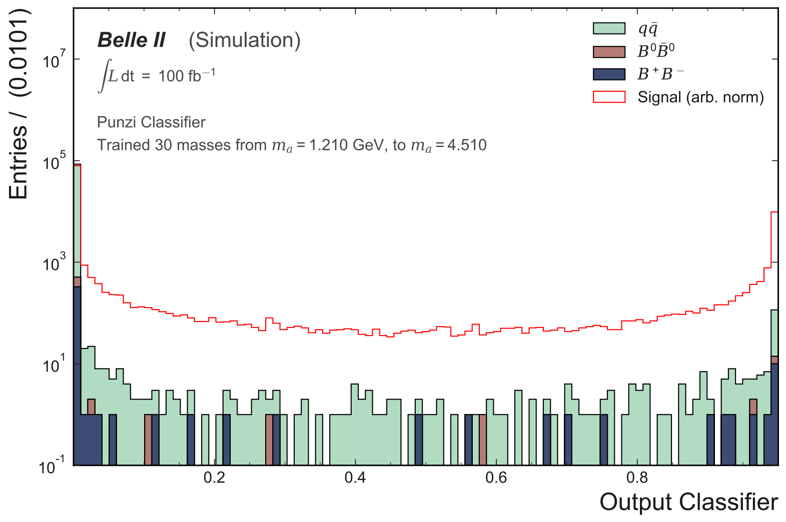


(a) BCE pretraining loss curves

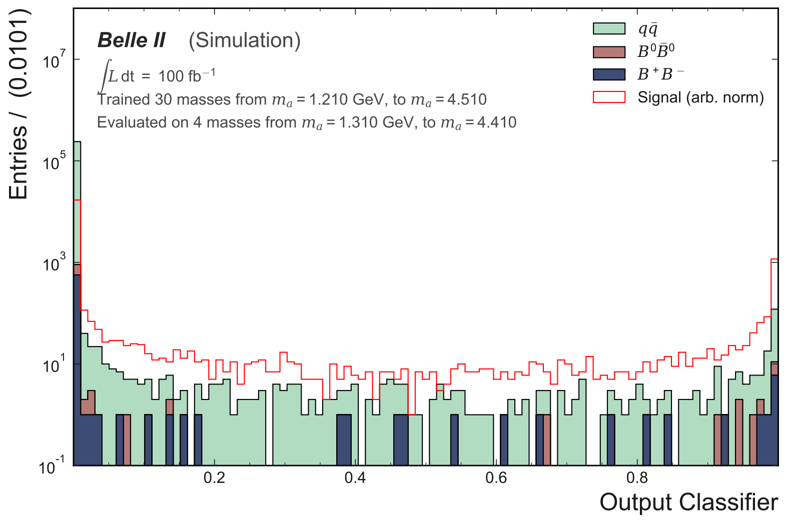


(b) Training Punzi-loss curves

Figure B.1.: These plots show the training and validation loss curves for the training on the additional set of variables with the small architecture. The loss during pretraining converges rapidly to a value of ca. 0.32 after 30 epochs. During the final training, the loss converges to approximately 350. The validation loss rises over the training loss after 50 epochs which can be evidence for overtraining.

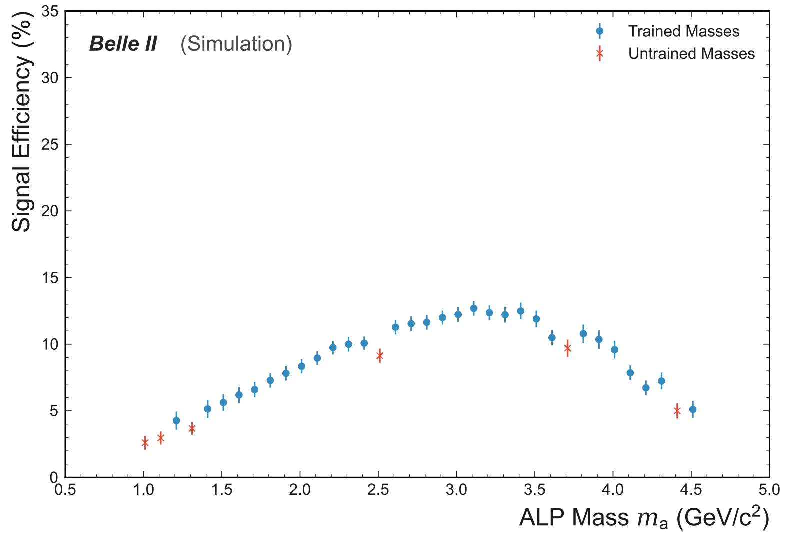


(a) Output for trained masses

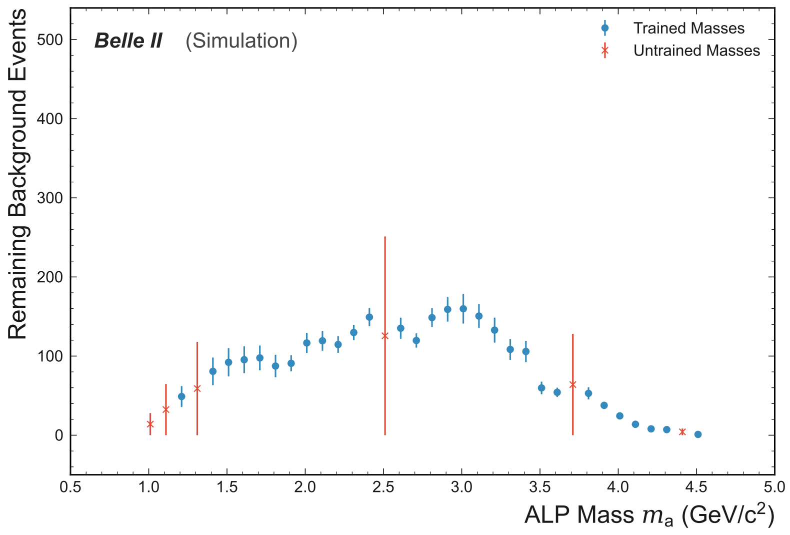


(b) Output for untrained masses

Figure B.2.: The plots show the output distribution of Punzinet in the original architecture for the trained masses (a) and untrained masses (b) with the additional set of parameters. The background histograms show very high numbers of background events classified as medium values, and a noticeable peak at a classifier of 1 which should be mostly populated by signal events. The classification of the signal events shows a high number of events classified with low background values which leads to a lower signal efficiency after applying the selection.



(a) Signal efficiency



(b) Background events

Figure B.3.: The plot shows the performance of the small Punzinet architecture on the additional variable set. To obtain the signal efficiency (a) and background events (b) the network trains 10 times and the results after a fixed selection at an output value of 0.8 are averaged. The signal efficiency this setup achieves averaged over all masses is $9 \pm 3\,\%$ at a background level of 80 ± 50. The histogram of the signal efficiency shows a maximum of 13 % for the medium masses with a descent down to 3 % for the outer masses. Untrained masses again do not align with the trained ones. The background events curve features a similar shape with a maximum of approximately 160 events and high deviations for the untrained masses.

Bibliography

[1] A. Heidelbach, "Sensitivity study in the search for $B^{\pm} \to K^{\pm}a$ (displaced $a \to \gamma\gamma$) decays at belle ii," Master's thesis, Karlsruhe Institute for Technology, Karlsruhe, 2022.

[2] F. Abudinén, M. Bertemes, S. Bilokin, M. Campajola, G. Casarosa, S. Cunliffe, L. Corona, M. De Nuccio, G. De Pietro, S. Dey, M. Eliachevitch, P. Feichtinger, T. Ferber, J. Gemmler, P. Goldenzweig, A. Gottmann, E. Graziani, H. Haigh, M. Hohmann, T. Humair, G. Inguglia, J. Kahn, T. Keck, I. Komarov, J. F. Krohn, T. Kuhr, S. Lacaprara, K. Lieret, R. Maiti, A. Martini, F. Meier, F. Metzner, M. Milesi, S. H. Park, M. Prim, C. Pulvermacher, M. Ritter, Y. Sato, C. Schwanda, W. Sutcliffe, U. Tamponi, F. Tenchini, P. Urquijo, L. Zani, R. Žlebčík, and A. Zupanc, "Punzi-loss:: a non-differentiable metric approximation for sensitivity optimisation in the search for new particles," *European Physical Journal C* **82** no. 2, (2022) .

[3] I. H. de la Cruz, "The Belle {II} experiment: fundamental physics at the flavor frontier," *Journal of Physics: Conference Series* **761** (2016) 12017. `https://doi.org/10.1088/1742-6596/761/1/012017`.

[4] D. Matvienko, "The Belle II experiment: status and physics program," *EPJ Web of Conferences* **191** (10, 2018) 02010. `https://www.epj-conferences.org/10.1051/epjconf/201819102010`.

[5] **Belle-II**, T. Abe *et al.*, "Belle II Technical Design Report," `arXiv:1011.0352 [physics.ins-det]`.

[6] T. Kuhr, C. Pulvermacher, M. Ritter, T. Hauth, and N. Braun, "The Belle II Core Software,". `http://arxiv.org/abs/1809.04299http://dx.doi.org/10.1007/s41781-018-0017-9`.

[7] G. C. Fox and S. Wolfram, "Event shapes in e+e- annihilation," *Nuclear Physics, Section B* **149** no. 3, (1979) 413–496.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics* **5** no. 4, (12, 1943) 115–133. `http://link.springer.com/10.1007/BF02478259`.

[9] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE* **78** no. 10, (1990) 1550–1560.

[10] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview,".

[11] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015) 1–15.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[13] N. Chukhrova and A. Johannssen, "Generalized two-tailed hypothesis testing for quantiles applied to the psychosocial status during the COVID-19 pandemic," *International Journal of Intelligent Systems* **36** no. 12, (12, 2021) 7412–7442. `https://onlinelibrary.wiley.com/doi/10.1002/int.22592`.

[14] G. Punzi, "Sensitivity of searches for new signals and its optimization," 2003. `https://arxiv.org/abs/physics/0308063`.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems* **32** no. NeurIPS, (2019) .

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature* **585** no. 7825, (9, 2020) 357–362.

[17] W. McKinney, "Data Structures for Statistical Computing in Python," pp. 56–61. 2010.

[18] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G.-L. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. de Miranda Cardoso, J. Reimer, J. Harrington, J. L. C. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, and Y. Vázquez-Baeza, "SciPy 1.0:

fundamental algorithms for scientific computing in Python," *Nature Methods* **17** no. 3, (3, 2020) 261–272.

[19] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," 2017.