AGH University of Science and Technology
Faculty of Physics and Applied Computer Science

Engineering thesis

Artur Kucia

field of study: **technical physics**

Track classification using machine learning techniques for LHCb experiment

Klasyfikacja śladów w eksperymencie LHCb przy pomocy uczenia maszynowego

Supervisor: **dr hab. inż. Tomasz Szumlak**

Kraków, January 2018

Aware of criminal liability for making untrue statements I declare that the following thesis was written personally by myself and that I did not use any sources but the ones mentioned in the dissertation itself.

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

.....

dr hab. inż. Tomasz Szumlak
Wydział Fizyki i Informatyki Stosowanej AGH
Katedra Oddziaływań i Detekcji Cząstek

Merytoryczna ocena pracy przez opiekuna:

Zastosowanie metod uczenia maszynowego zdobyło bardzo dużą popularność w dziedzinie fizyki cząstek elementarnych. Dotyczy to zwłaszcza systemów działających w czasie rzeczywistym takich jak układy wyzwalania przypadków (ang. trigger). Praca, której Autorem jest Pan Artur Kucia związana jest właśnie z poprawą wydajności algorytmu do rekonstrukcji śladów cząstek długo-życiowych w eksperymencie LHCb (ang. Large Hadron Collider beauty), działającym przy akceleratorze LHC (ang. Large Hadron Collider). Głównym celem przedstawionego projektu była analiza możliwości wykorzystania sieci neuronowych do klasyfikacji śladów rekonstruowanych przez detektor LHCb. Główne wyzwania, które postawił przed sobą Autor dotyczyły opracowania odpowiedniego modelu (liczba neuronów oraz warstw sieci), wytrenowania go tak aby klasyfikował ślady z wysoka wydajnością, stworzenie kodu który mógł zostać użyty w oficjalnym oprogramowaniu eksperymentu LHCb i wreszcie analiza czasu wykonania, który ma decydujące znaczenie przy ocenie możliwości zastosowania sieci neuronowych w układzie wyzwalania.

Praca napisana jest w języku angielskim. Składa się z pięciu rozdziałów i jest zakończona podsumowaniem. W pierwszej części Autor wprowadza czytelnika do zagadnień związanych z fizyką wysokich energii, omawia krótko detektor LHCb i przedstawia zagadnienie rekonstrukcji śladów. Następny rozdział stanowi bardzo dobre wprowadzenie (na poziomie dydaktycznym) do zagadnień uczenia maszynowego ze szczególnym naciskiem na techniki wykorzystujące sztuczne sieci neuronowe. W rozdziale tym znajdziemy również opis klasyfikatora binarnego oraz sposobów oceny jego działania. Rozdział czwarty zawiera dokładny opis procesu trenowania sieci, badania jakości jej odpowiedzi i integracji z oprogramowaniem LHCb. Rozdział piąty zawiera analizę wyników pod kątem fizyki rozpadu cząstek długożyciowych K_s^0 oraz Λ^0, który pozwala na ostateczną ocenę przydatności wykorzystanego algorytmu do klasyfikacji.

Uzyskane wyniki pokazują, że sztuczne sieci neuronowe mogą zostać użyte do klasyfikacji śladów cząstek. Autor pokazał również, że odpowiednio przygotowany kod reprezentujący odpowiedź wytrenowanej sieci może być częścią systemu czasu rzeczywistego. Końcowy model działał z dokładnością wynoszącą 88% natomiast powierzchnia pod krzywą ROC (ang. Receiver Operating Characteristic) wyniosła 94%.

W podsumowaniu, chciałbym podkreślić duże zaangażowanie oraz samodzielność Autora. Sztuczna sieć neuronowa, została użyta po raz pierwszy do rekonstrukcji śladów w eksperymencie LHCb. Napisana praca stanowi świetne źródło informacji dla osób zainteresowanych zastosowaniem technik uczenia maszynowego. Dodatkowo, Autor będzie kontynuować prace nad klasyfikacją śladów dla zmodernizowanego eksperymentu LHCb, który będzie zbierać dane po roku 2020.

Końcowa ocena pracy przez opiekuna: 5.0

Data: 22.1.2018r. Podpis: .....

Skala ocen: 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostateczna, 2.0 – niedostateczna

dr inż. Bartłomiej Rachwał
Wydział Fizyki i Informatyki Stosowanej AGH
Katedra Oddziaływań i Detekcji Cząstek

Merytoryczna ocena pracy przez recenzenta:

Projekt inżynierski Pana Artura Kucia dotyczył opracowania nowego narzędzia do klasyfikacji rekonstruowanych śladów cząstek długożyciowych w eksperymencie LHCb. Zadanie tego typu zawiera niezwykle istotną z punktu widzenia działania eksperymentu motywację fizyczną - wydajność rekonstrukcji śladów oraz stopień redukcji śladów fałszywych bezpośrednio przekłada się na jakość zebranych przez eksperyment danych.

Eksperyment LHCb poświęcony jest badaniu szczególnych zjawisk w fizyce wysokich energii. Jest on umiejscowiony przy Wielkim Zderzaczu Hadronów (LHC) w europejskim ośrodku badań jądrowych, CERN pod Genewą. Dzięki specyficznej konstrukcji spektrometru LHCb, jak również bardzo dobrej wydajności zbierania danych, detektor ten nadaje się do szerokiego zakresu badań fizyki "do przodu", o czym traktuje Autor pracy w zamieszczonym wprowadzeniu. W zwięzły i intuicyjny sposób wprowadza czytelnika do przedmiotu podjętego przez siebie zadania w ramach pracy inżynierskiej, uwzględniając szeroko pojęte uczenie maszynowe oraz algorytmy sztucznych sieci neuronowych.

Struktura przedłożonej pracy inżynierskiej jest bardzo logiczna, w kolejności zawiera niezbędne opisy pozwalające zrozumieć główną część pracy. I tak, w rozdziale o metodologii realizowanego zadania, bardzo skrupulatnie zdefiniowane są zestawy danych, zmienne uczące, oraz sam proces uczenia sieci.

Istotnym z punktu widzenia społeczności eksperymentu są uzyskane wyniki oraz integracja całej procedury z oprogramowaniem LHCb, co również zostało opisane w pracy. Na dowód poprawności uzyskanych wyników, Autor zamieścił również rozdział dedykowany wydajności fizycznej wytrenowanego klasyfikatora oraz jego wpływ na konkretne pomiary masy niezmienniczej rekonstruowanych cząstek K_0^S oraz Lambda^0.

Autor pracy bezspornie poradził sobie ze stroną techniczną projektu, wykazując się przy tym bardzo dobrymi umiejętnościami z programowania! Zapoznał się z oficjalnym oprogramowaniem eksperymentu LHCb i na tej podstawie zdołał zaimplementować główny cel pracy do środowiska oprogramowania LHCb.

Podsumowując, przedstawiona praca od strony merytorycznej jest bardzo dobra. Napisana w języku angielskim, stanowi bez wątpienia pozycję po którą członkowie grupy roboczej w ramach współpracy LHCb mogą sięgać jako dokumentację praktycznego i wydajnego narzędzia jakim jest klasyfikator śladów cząstek w procesie ich rekonstrukcji w detektorze!
Końcowa ocena pracy przez recenzenta: 5.0

Data: 24.1.2018r Podpis: .....

Skala ocen: 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostateczna, 2.0 – niedostateczna

Abstract

Efficient track reconstruction of charged particles originating from proton-proton collisions at Large Hadron Collider (LHC) is a challenging task. The enormous number of "hits" produced in the various tracking detectors often leads to poor performance when tackled by conventional statistical methods. This thesis describes the development of a new classifier that is a vital component of the algorithm used for long-lived particles reconstruction at LHCb experiment. The Author successfully implemented a novel approach employing a neural network algorithm trained using datasets from Monte Carlo simulations. The resulting classifier decreases the track reconstruction efficiency by 1% but reduces a ghost fraction by 7%. The algorithm execution times are similar to previously used approaches. The work methodology developed by this thesis can be the base for future projects involving the neural networks models in LHCb software.

Abstract

Wydajna rekonstrukcja śladów naładowanych cząstek pochodzących ze zderzeń protonów w Wielkim Zderzaczu Hadronów jest trudnym zadaniem. Z powodu ogromnej liczby śladów wytworzonych w różnych detektorach użycie konwencjonalnych metod statystyczych często prowadzi do niezadowalających rezultatów. Niniejsza praca dyplomowa opisuje stworzenie nowego klasyfikatora, który jest niezbędnym elementem algorytmu używanego do rekonstrukcji śladów cząstek długożyciowych w eksprymencie LHCb. Autor z sukcesem wdrożył nowatorską technikę wykorzystującą sieć neuronową wytrenowaną na danych pochodzących z symulacji Monte Carlo. Powstały klasyfikator obniża wydajność rekonstrukcji śladów o 1%, ale redukuje ilość śladów fałszywych o 7%. Czas wykonywania algorytmu jest porównywalny z wcześniej używanym rozwiązaniem. Przedstawiona metodologia może być podstawą przyszłych projektów wykorzystujących sieci neuronowe w oprogramowaniu LHCb.

Contents

1 Introduction

The "Large Hadron Collider beauty" (LHCb) is one of the four big experiments located at CERN near the city of Geneva at the French-Switzerland border. The detector has a different construction from all other experiments at LHC - it is a single-arm spectrometer with a forward angular coverage from approximately 10 mrad to 300 (250) mrad in the bending (non-bending) plane[1]. It was designed to study the interactions of hadrons consisting of b and $\bar{b}$ quarks. These are thought to be the source of CP violation which cannot be fully explained by the Standard Model of particle physics[2] and can point researchers to the new phenomena, so-called *New Physics*

In order to study the physics of interacting particles, all products of proton-proton collisions have to be identified and their properties measured. This is done by large hybrid detectors comprising of many complex sub-systems. The project focuses on track reconstruction of decay products of long-lived particles called the Downstream tracks. Such particles provide limited information to the system because their mother particles are neutral and cannot produce hits in the detectors located near the beam crossing point. The major problem with reconstructing such tracks is that the hits left by them are nearly indistinguishable from all other collision products. The track reconstruction algorithms consist of 3 stages: pattern recognition, where hits in the detectors' modules are selected as track candidates, fitting, which estimates the track parameters and clone killing which removes the duplicates from fitted tracks.

Artificial neural networks are a flexible and amazingly powerful class of machine learning algorithms. In the recent years, various kinds of neural networks have achieved state-of-the-art performance on problems like object recognition [3], language translation and have conquered the world's champion in the game of Go [4]. These tasks were considered to stay beyond the bounds of computers' capabilities at least for a couple more decades. Although the neural networks have been invented in the middle of 20th century, advances only became possible, when large datasets describing various kinds of problems became available. In this project, the neural network model is trained to provide a binary classification of good and bad T-Seed track segments used for reconstruction of long-lived particles by the algorithm called *PatLongLivedTracking*[5]. The training data were obtained from the Monte Carlo simulations.

The developed software needed to be integrated within existing LHCb core software - the Gaudi framework[6]. To be useful for future applications the algorithm has to be implemented in C++ language and besides being accurate, must be as computationally efficient as possible. This poses the restriction on the complexity level of the used model.

The thesis is structured as follows. Section 2 introduces the concepts and definitions used for the Downstream Tracking problem and Section 3 for the machine learning algorithms. Part 4 contains the detailed description of the new T-Seed Classifier development process and its implementation within existing software framework. Section 5 presents the evaluation of algorithm's performance on a simulated and real collision data. Finally, part

6 gives the conclusions.

2 Experimental background

2.1 The LHCb experiment

The LHCb detector consists of many submodules, each designed to perform a specific function. Figure 1 depicts the LHCb detector schematic and its key submodules:[1]

- **VELO** - Vertex Locator system.
  Being the closest to the interaction point of proton beams, it provides precise measurements of primary and secondary vertices.
- **RICH1** and **RICH2** - Ring Imagining Cherenkov counters
  Identify the traversing hadrons based on the measurements of their emitted Cherenkov radiation.
- **TT** and **T1** - **T3** - Tracker Turicensis and three Tracking stations.
  The tracking system responsible for the reconstruction of trajectories of charged particles and momentum measurements.
- **Magnet**
  Bends the trajectories of charged particles, which allows for the calculation of the momentum from the track curvature.
- **ECAL** - Electromagnetic Calorimeter.
  Measures the energy and positions of electrons and photons.
- **HCAL** - Hadron Calorimeter.
  Measures the energy and position of protons, neutrons, pions and muons.
- **M1** to **M5** - Muon Chambers.
  Identify the muons and measures their momenta.

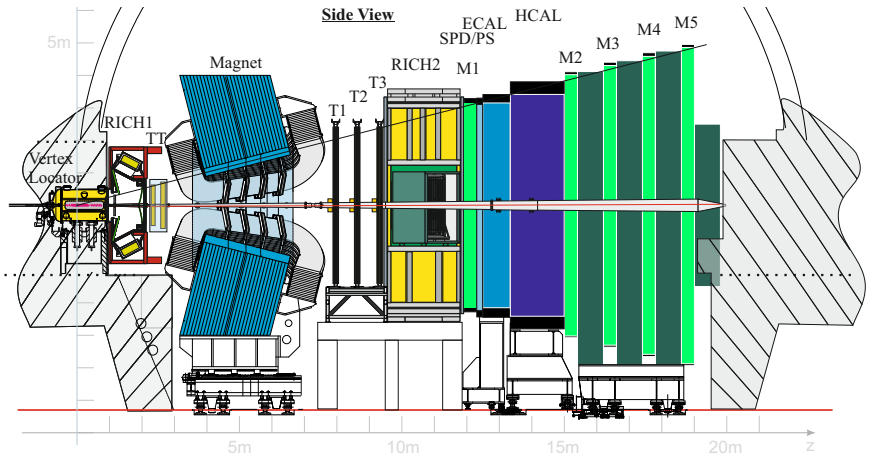

Figure 1: Schematic layout of the LHCb detector[7]. The right-handed coordinate system is used with z along the beam axis into the detector, y vertical and x in the horizontal direction.

2.2 The LHCb Software

The LHCb software is organised to maximise the efficiency of processing and storing large amounts of data. Every step in the pipeline processes data event-by-event, receiving it from the previous application and preparing new data for the next (Figure 2).

There are two sources of event data:

- Real collisions - Events recorded by the detector hardware.
- Simulated events - Created by Monte Carlo (MC) generators coordinated by the *Gauss*[8] and converted to resemble real data by the *Boole*[9] application (**Particle Simulation** and **Digitisation**).

Irrespective of the source of the event, data is processed by the *Moore*[10], which filters out uninteresting events (**Trigger**). Trajectories of particles in events left by the trigger step are created by the *Brunel*[11] (**Reconstruction**), and the *DaVinci*[12] module performs further data filtering (**Stripping**). Finally, events can be stored and made available to physicists for analysis.

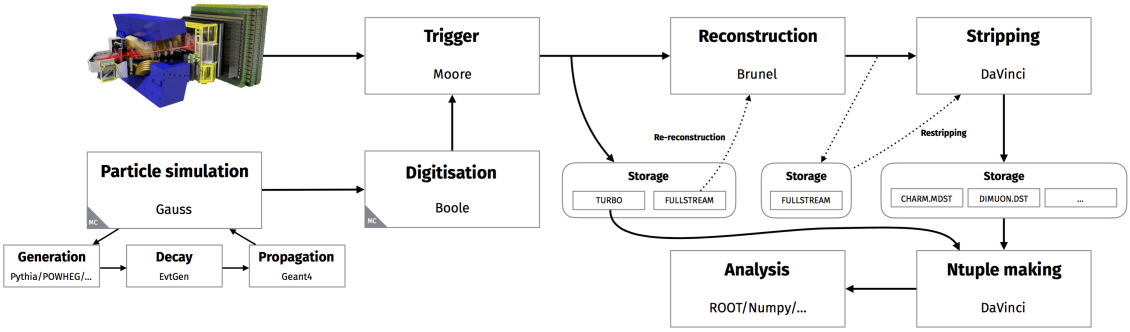

Figure 2: The LHCb run 2 data flow [13].

2.3 Track types

Particles produced at the beam intersection point pass through the detectors modules. The clusters of charge left on their path are recorded by various sub-detectors and form *hits*. These hits have to be then connected to reconstruct the original particle track. Tracks provide information about estimated momentum and primary vertex position, which in turn are used for particle identification.

Fitted tracks are represented by the set of straight line segments at different z positions called the *states*[14]. Every state forms a 5-dimensional vector which contains the state's position in the x-y plane, the slope with respect to the z-axis and the particle's track curvature (Eq. 1).

$$\vec{x} = \begin{pmatrix} x \\ y \\ tx \\ ty \\ \frac{p}{q} \end{pmatrix}, \; tx \equiv \frac{\partial x}{\partial z}, \; ty \equiv \frac{\partial y}{\partial z} \tag{1}$$

Tracks reconstructed in the LHCb detector are assigned to one of the following categories based on the location of their hits[15] (Fig. 3):

- **Long tracks** - have hits in all detector components which make them the most useful tracks for physics analysis.
- **Upstream tracks** - have hits in the VELO and TT, but are bend out of the detector by the magnetic field before reaching the T stations.
- **Downstream tracks** - have hits only in TT and T stations. They are left by the decay products of long-lived neutral particles.
- **VELO tracks** - have hits only in the VELO. They are used to find the primary vertex's position and as input to Long and Upstream tracking reconstruction algorithms.
- **T tracks** - have hits only in the T stations. They are used as input to Long and Downstream tracking algorithms.

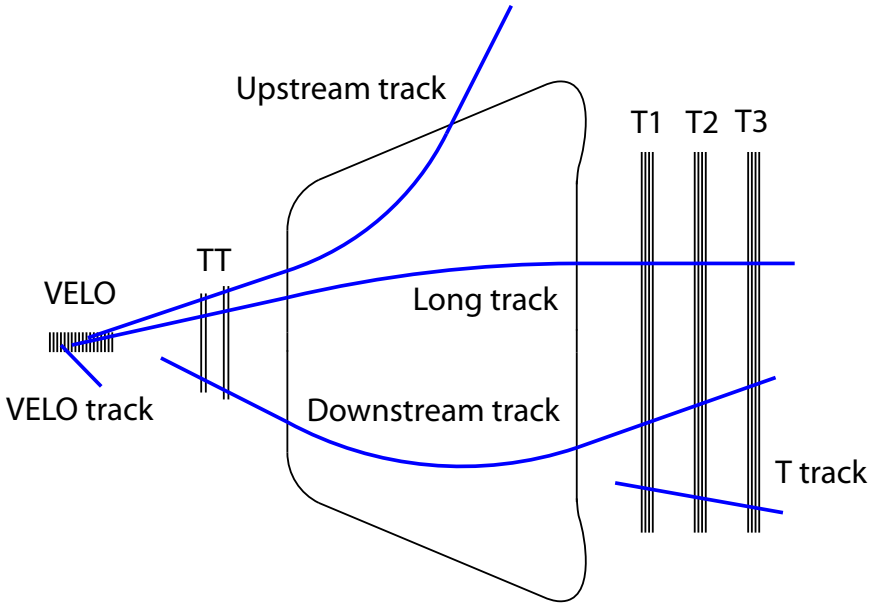

Figure 3: LHCb sub-detectors schematic with different track types[16].

2.4 Downstream tracks reconstruction

The current implementation of Downstream track reconstruction algorithm is performed by the *PatLongLivedTracking* algorithm, which could be summarised by three steps:

1. **Filtering**
   Selecting the T-tracks in the T-stations and assigning them to particle track candidates.
2. **Track fitting**
   Extending the track states for every track candidate left from the previous step by propagating them through the magnetic field in search for matching TT hits.
3. **Evaluation of track candidates**
   Choosing the best tracks with a multivariate classifier.

This thesis focuses on improvements of its initial filtering module - *T-seed classifier*.

From all T-track seeds (referred further as seeds or T-seeds), which are the input of PatLongLivedTracking algorithm, only a fraction of seeds corresponds to real Downstream tracks. To reduce the complexity of a problem and for further steps and improve the overall quality of fitted tracks, a first stage of the algorithm filters out bad (or ghost) candidates for reconstruction.

Currently PatLongLivedTracking uses the XGBoost[17] classifier. Due to the limitations of LHCb High Level Trigger computation resources the classifier does not operate on continuous values as inputs, but instead, it uses a binned version (bonsai BDT[18]). The input variables and model decisions are mapped to limited discrete space and stored as a lookup table[5]. The table size growths rapidly with a number of used input variables and it is difficult to meet required memory criteria. What is more, the binning procedure reduces the classification performance of the model.

2.5 Figures of merit

- **Reconstruction efficiency**[15]
  For the Downstream tracking problem, the MC particle is said to be *reconstructible* if it is reconstructible as T track (it has at least one hit in each plane of every T station) and has at least one hit in both planes of TT. The MC particle is said to be *reconstructed* if at least 70% of the hits on the track come from this particle. Besides the track is required to not originate from the electron. The *reconstruction efficiency* (Eq. 2) is a fraction of correctly reconstructed Downstream tracks within all reconstructible Downstream tracks. Naturally, higher values correspond to better performance.

$$\epsilon_{rec} = \frac{number\ of\ reconstructible\ and\ reconstructed\ tracks}{number\ of\ reconstructible\ tracks} \tag{2}$$

- **Ghost rate**[15]
  A track is said to be a *ghost* if there are no MC particles associated with it. The *ghost rate* is the fraction of such tracks within all tracks (Eq. 3). A Lower value means better algorithm performance.

$$ghost\ fraction = \frac{number\ of\ ghost\ tracks}{number\ of\ tracks} \tag{3}$$

- **Execution time**

  The algorithm can be run by the LHCb Trigger. Therefore it is required, that the CPU execution time is taken into consideration and minimised if possible.

- **Signal significance**

  The final performance metric, which must be considered, is the direct influence on the particle detection chance. The signal significance can be defined as the number of standard deviations of background distribution, to which the signal corresponds.[19]

$$significance = \frac{signal\ events}{\sqrt{signal\ events + background\ events}} \tag{4}$$

3 Machine learning

3.1 Solving problems with Machine Learning

> We have at our command computers with adequate data-handling ability and with sufficient, computational specs to make use of machine-learning techniques, but our knowledge of the basic principles of these techniques is still rudimentary. Lacking such knowledge, it is necessary to specify methods of problem solution in minute and exact detail a time-consuming and costly procedure. Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort.
> *Arthur Samuel, 1959* [20]

Based on this statement of Arthur Samuel, an artificial intelligence pioneer, machine learning can be described as providing the computers with the ability to solve problems without being explicitly instructed.

This work focuses on *supervised learning*, where the algorithm is provided with example data and problem solutions - the *label* or *target*. Most common examples of supervised learning algorithm are linear regression and classification.

Every machine learning model has its *parameters* - a set of variables, which are changed during training based on the evaluated algorithm's performance. On the contrary, the parameters which need to be specified before the training begins are called *hyperparameters*. Referring again to the regression example, a hyperparameter would be a degree of the fitted polynomial, whereas its coefficients would be the model's parameters.

The algorithm *training* refers to optimisation procedure of changing the models' parameters in such a way, which minimise a chosen target evaluation metric. This metric is called the *loss function*. It is required, that the loss function provides a continuous-valued score for each example[21]. One can say, that an algorithm is *learning* to solve a problem.

In the test phase, the model's ability to change its parameters is disabled, and it is used only to provide predictions on previously unseen data.

The difference between training and testing performance is called *generalisation error* (Figure 4). The generalisation error can be decomposed into *bias* and *variance*[22]. Highly

biased models (low variance) give consistent predictions but are highly inaccurate. Their complexity is not sufficient for a given task. Such behaviour is called *underfitting*. Models with high variance (low bias) give accurate predictions on training dataset but have a poor performance on a test dataset. This is called *overfitting*. Large models with millions of parameters are able to "memorise" internally entire datasets.

Various *regularisation* techniques introduce restrictions on the model complexity to combat the overfitting. Adding the sum of the model parameters to the loss function is a simple example of such technique. It effectively forces model to use smaller parameter values. A proper testing procedure is crucial for finding the optimal model capacity which assures the best model generalisation and prevents overfitting.

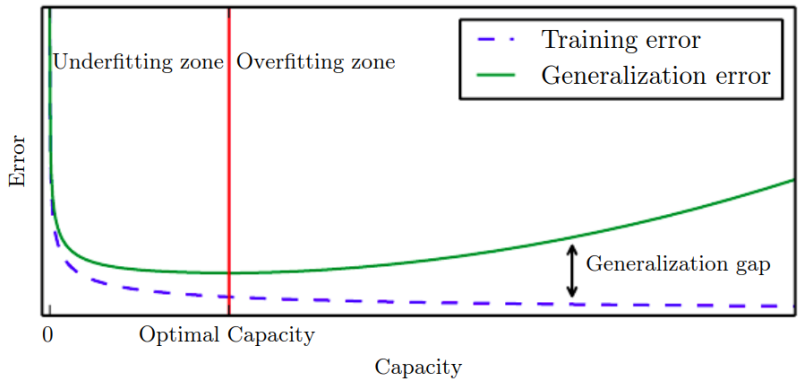

Figure 4: An example which illustrates bias-variance trade-off. The model with low capacity is highly biased. The model with too much capacity suffers from too much variance. Optimal capacity has similar scores for both training and testing errors.[21]

3.2 Artificial Neural Networks

The simplest building block of the network is called *the neuron*. Its behaviour is inspired by the simplified biological neuron model[23]. Every neuron performs calculations on all of its input values and outputs a single value. The schematic of this computations is shown in Figures 5 and 6. The model of the layer of neurons can be written in the form of the matrix equation (Eq.5).

$$\begin{pmatrix} h_1 & h_2 & \cdots & h_j \end{pmatrix} = f(\begin{pmatrix} x_1 & x_2 & \dots & x_i \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1j} \\ w_{21} & w_{22} & \cdots & w_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & \cdots & w_{ij} \end{pmatrix} + \begin{pmatrix} b_1 & b_2 & \dots & b_j \end{pmatrix}) \tag{5}$$

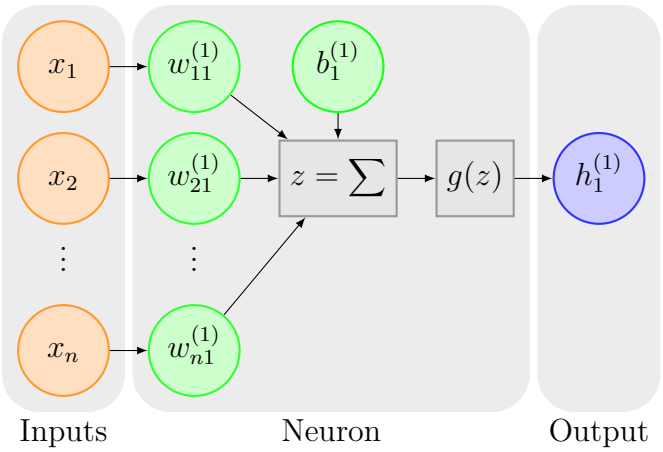

Figure 5: Schematic of a single neuron unit.

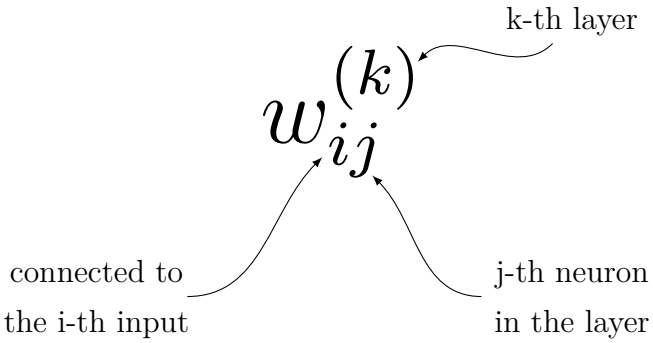

Figure 6: Weight description convention.

The input vector $\mathbf{x}$ is multiplied by the matrix of internal neurons' weights $\mathbf{W}$. Every column of the matrix corresponds to a different neuron in the layer. Next, the bias term is added. This affine transformation of inputs is commonly referred to as the *dense* layer. Finally, the activation function is applied element-wise to the vector of outputs.

The activation function in a neuron is essential to introduce nonlinearities in the model. Without it, stacking multiple dense layers of neurons on top of each other would have no better performance than single, larger layer. Examples of commonly used activation functions are presented in Figure 7.

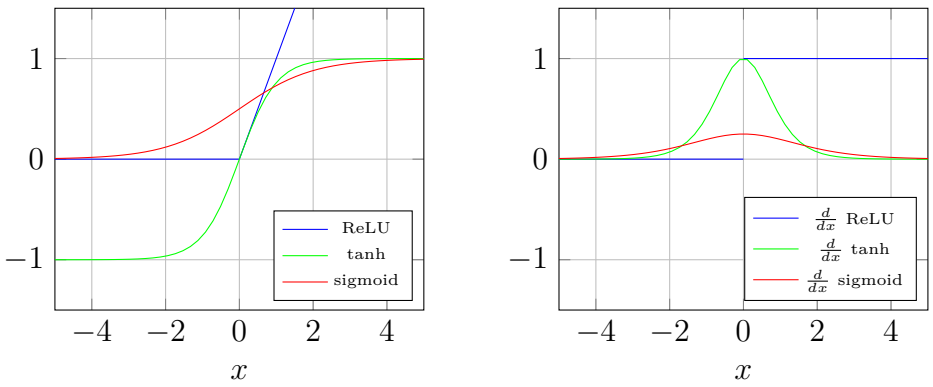

Figure 7: Commonly used neuron activation functions(left) and their derivatives (right).

A useful technique that often increases network performance is *batch normalisation* [24]. In a nutshell, it transforms the activations of the layer to form a unit Gaussian distribution, which prevents the individual neurons output values from dominating over other in a layer.

The most straightforward neural network model is created by connecting the layer's outputs to the next layers' inputs (Figure 8). Every layer of the network creates new features that are used by the following layer. The outputs of the last layer are interpreted as the network's predictions.

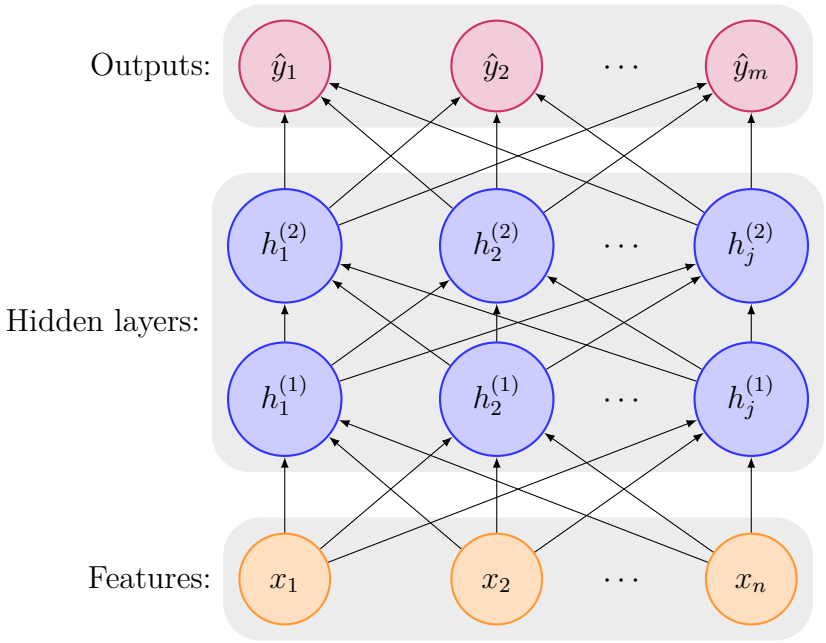

Figure 8: Diagram of a neural network model with two hidden dense layers.

Neural network training involves the iterative change of weights in such a way as to minimise the loss function. Initial weights are chosen randomly. The weight update rules are dictated by the *optimiser*. The simplest approach is to use plain gradient descent, but in practice, its variation called *stochastic gradient descent* achieves better results. Instead of calculating the gradient on an entire dataset, the examples are randomly split into smaller *batches*. This approach introduces noise to the estimated weight updates, which helps to avoid the local minima of the loss function[25]. Furthermore splitting data into smaller parts allows for the more efficient use of resources on specific computer architectures, such as GPUs. The exact batch size depends on the problem and available computing resources but typically ranges from few to a few thousand examples. In the recent years, many variations of stochastic gradient descent have been developed. They depend on many heuristics which speed up the training process[26]. Efficient training of a neural network is possible thanks to the *backpropagation* algorithm[21]. It applies the chain rule of differentiation to produce partial derivatives of the loss function. The derivatives over all of the network weights are stored, in their symbolic form, in a computational graph. The exact details of the backpropagation algorithm are beyond the scope of this

thesis.

3.3 Binary classification with neural networks

The binary classification can be viewed as a task learning a Bernoulli distribution over y conditioned at $\mathbf{x}$. We can denote the true signal as 1 and background as 0. The neural network outputs the value of function $g_s(\mathbf{x})$ which is the probability of the random variable y taking the value 1 given that the input to the model was $\mathbf{x}$ (Eq. 6).

$$\hat{y} = g_s(\mathbf{x}) = P(y = 1|\mathbf{x}) \tag{6}$$

The output of the network, $\hat{y}$, is required to lie in the interval $[0, 1]$. This can be achieved by using a sigmoid (Figure 7) as an activation function of the output neuron.

The loss function usually applied in binary classification problem is *mean cross entropy loss*[27] defined as Eq. 7.

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)] \tag{7}$$

3.4 Evaluating performance of binary classifiers

The following metrics provide a better description of model performance, then the loss function. They are used to measure the trained model quality and to compare different models.

- **Accuracy**

  The accuracy of the classifier is one of the most common evaluation metric (Eq. 8):

  $$accuracy = \frac{number\ of\ correctly\ classified\ examples}{number\ of\ all\ examples} \tag{8}$$

  Scores depend on the chosen decision threshold (th) and the information about the confidence of the model is not taken into account. Therefore this metric does not measure the certainty of predictions.

- **Confusion Matrix**

  A confusion matrix for binary classification is a table with two rows and two columns. Each column corresponds to a predicted class and each row to ground truth class. The examples which were correctly classified lie on the main diagonal - true negatives (TN) and true positives (TP). The errors are located on the antidiagonal - false positives (FP, type I errors), and false negatives (FN, type II errors). A visualisation of example confusion matrix is shown in Figure 9. A perfect classifier would have only true positives and true negatives, so its confusion matrix would have nonzero values only on its main diagonal[27]. A normalised confusion matrix has its rows divided by the number of examples in each class[28]. Confusion matrix allows choosing optimal th value which minimises type I or type II errors.

		Predicted label	
		false	true
Actual label	false	TN	FP
	true	FN	TP

		Predicted label	
		false	true
Actual label	false	0.85	0.15
	true	0.05	0.95

Figure 9: Performance measures depicted by the confusion matrix (left). Example of a normalised confusion matrix (right).

- **Receiver operating characteristic and ROC AUC score**
  Receiver operating characteristic (ROC) curve is a plot of true positive rate as the function of false positive rate[29]. The worst possible classifier, one that does not provide any discrimination between classes, forms a straight line at 45° angle on the plot. Steeper and closer to upper left corner curves indicate the better certainty of predictions. The ideal model, which maximises the true positive rate and minimises the false positive rate forms a curve with a right angle. An example of ROC curve is shown in Figure 10. An easy way to compare different models is to measure the total area under the ROC - the AUC score.

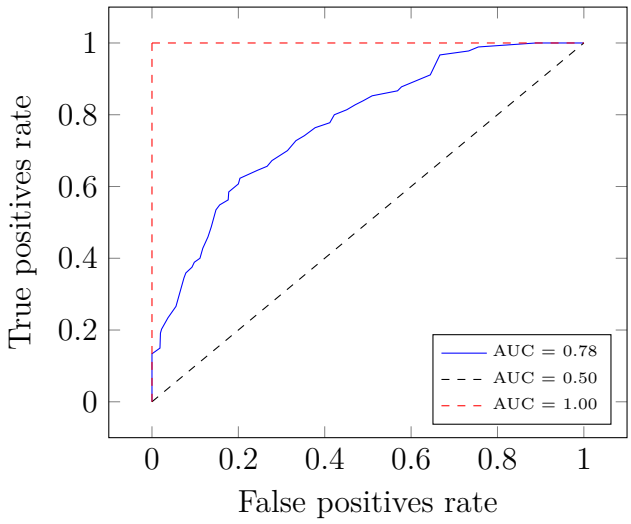

Figure 10: An example of receiver operating characteristic curve. Worst (dashed black line), mediocre (solid blue line) and perfect classifier (dashed red line).

4 Methodology

4.1 Dataset analysis

The data for the project was provided in the form of database (called nTuple) containing reconstructed T-track segments (seeds) from the $B^0 \to J/\psi K_s$ decays generated by

the LHCb MC simulations. Each seed example had been labelled based on the information produced in respective tracking detectors (TT and T stations) and association with true particles (so-called MC truth information). The positive examples made for 53% of the training set (Figure 11). The class imbalance was minimal and did not require undersampling or oversampling. The losses of negative examples were scaled by a factor of 1.084 and positive examples by 0.928 to equalise the importance of both. Available input features are summarised in Table 1 and their histograms are shown on Figure 12.

Figure 11: Target class balance. The value 1 denotes a true seed and 0 ghost.

Table 1: Description of original features from the nTuple database.

feature	values	description
seed_chi2PerDoF	continous	χ^2 per degree of freedom of track
seed_p	continous	track momentum
seed_pt	continous	transversal track momentum
seed_nLHCbIDs	discrete	number of hits associated to the T-Seed
seed_nbIT	discrete	number of IT hits associated to the T-Seed
seed_nLayers	discrete	number of used T station layers
seed_x	continous	x coordinate T-Seed projection onto first plane of T-Station
seed_y	continous	y coordinate T-Seed projection onto first plane of T-Station
seed_tx	continous	x coordinate of the T-Seed slope
seed_ty	continous	y coordinate of the T-Seed slope

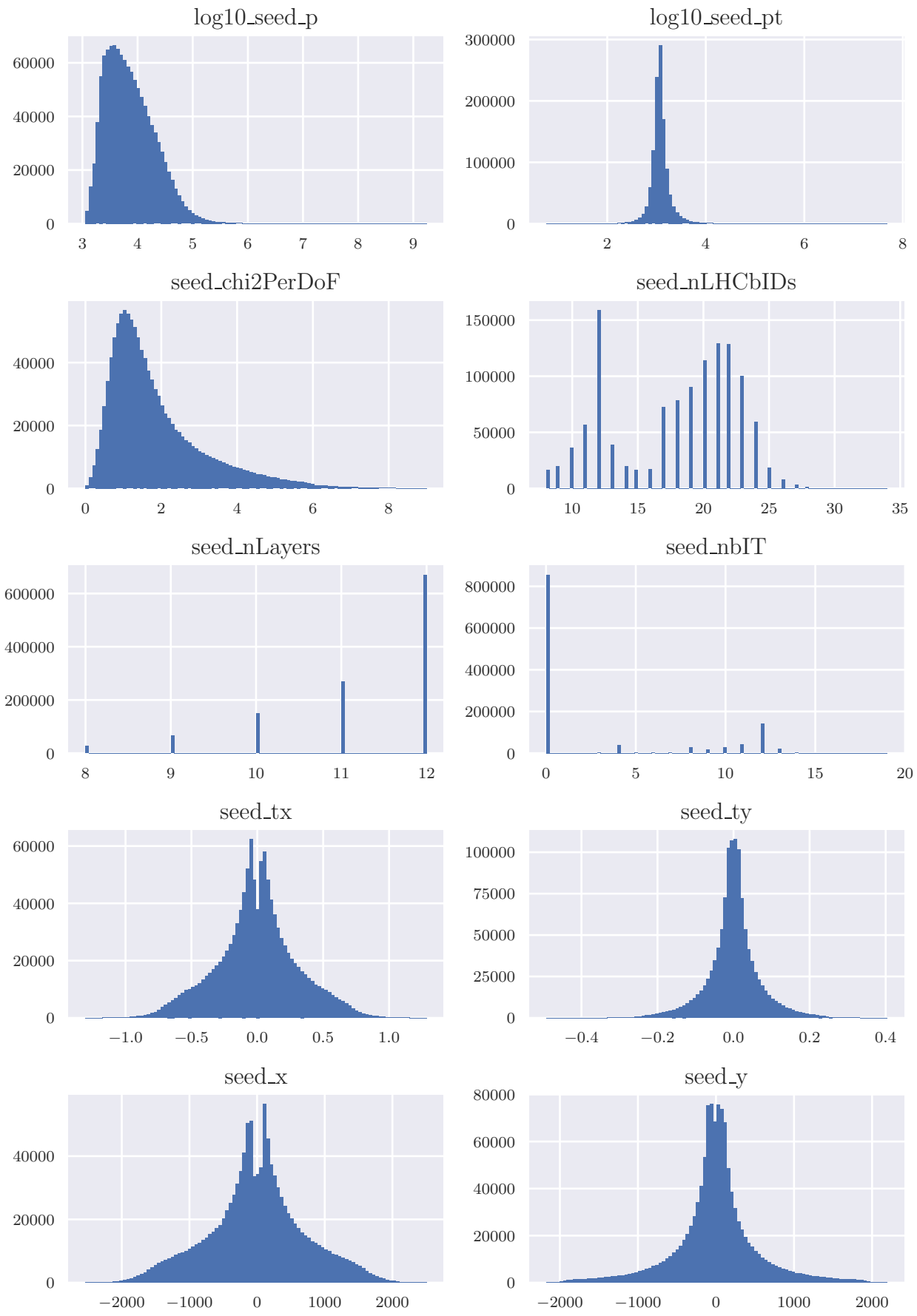

Figure 12: Histograms of original features from the nTuple database.

To to provide more knowledge about the physics of the process, new features, described in Table 2, were added to the dataset.

Table 2: Description of new features added as the network inputs.

feature	values	description
seed_pr	continous	pseudo rapidity of the T-Seed
seed_r	continous	radius of T-Seed projection onto first plane of T-Station
seed_phi	continous	angle of T-Seed projection onto first plane of T-Station

All discrete variables were rejected from the dataset. It was found that they did not help the classification algorithms in providing better predictions.

Proper training of the neural network requires scaling the inputs to have zero mean and unit variance, which equalises the importance of different inputs[30]. The features' values were shifted by the computed mean and scaled by the inverse of the standard deviation (Table 3). Additionally, the square root of 'seed_chi2PerDoF' was computed, and 'seed_p' and 'seed_pt' were log transformed before scaling. This procedure created similarly scaled features and enabled efficient training of the model (Figure 13).

The available dataset was randomly divided into two sets in the proportion of 7:3. The second, smaller, set was held out for final evaluation. The larger set was divided again: 80% of examples was chosen as training set and remaining 20% as the validation set (Table 4). Both were used in experiments - the training set was used to obtain normalisation parameters and train the models and validation set was used to evaluate the selection of hyperparameters and to monitor possible overfitting.

Table 3: Normalisation parameters estimated on the training dataset

feature	shift	scale
sqrt_seed_chi2PerDoFS	-1.29841	2.32293
log10_seed_p	-3.87308	2.21463
log10_seed_pt	-3.07839	4.61096
seed_x	-0.19296	0.00141
seed_y	4.77015	0.00192
seed_tx	0.00249	3.26001
seed_ty	0.00061	14.81124
seed_pr	-0.23735	5.42525
seed_r	-712.28415	0.00193
seed_angle	0.00059	1.34991

Table 4: Training, validation and test dataset sizes

set	size	% total
train	1 352 512	56
validate	338 128	14
test	724 560	30
total	2 415 200	100

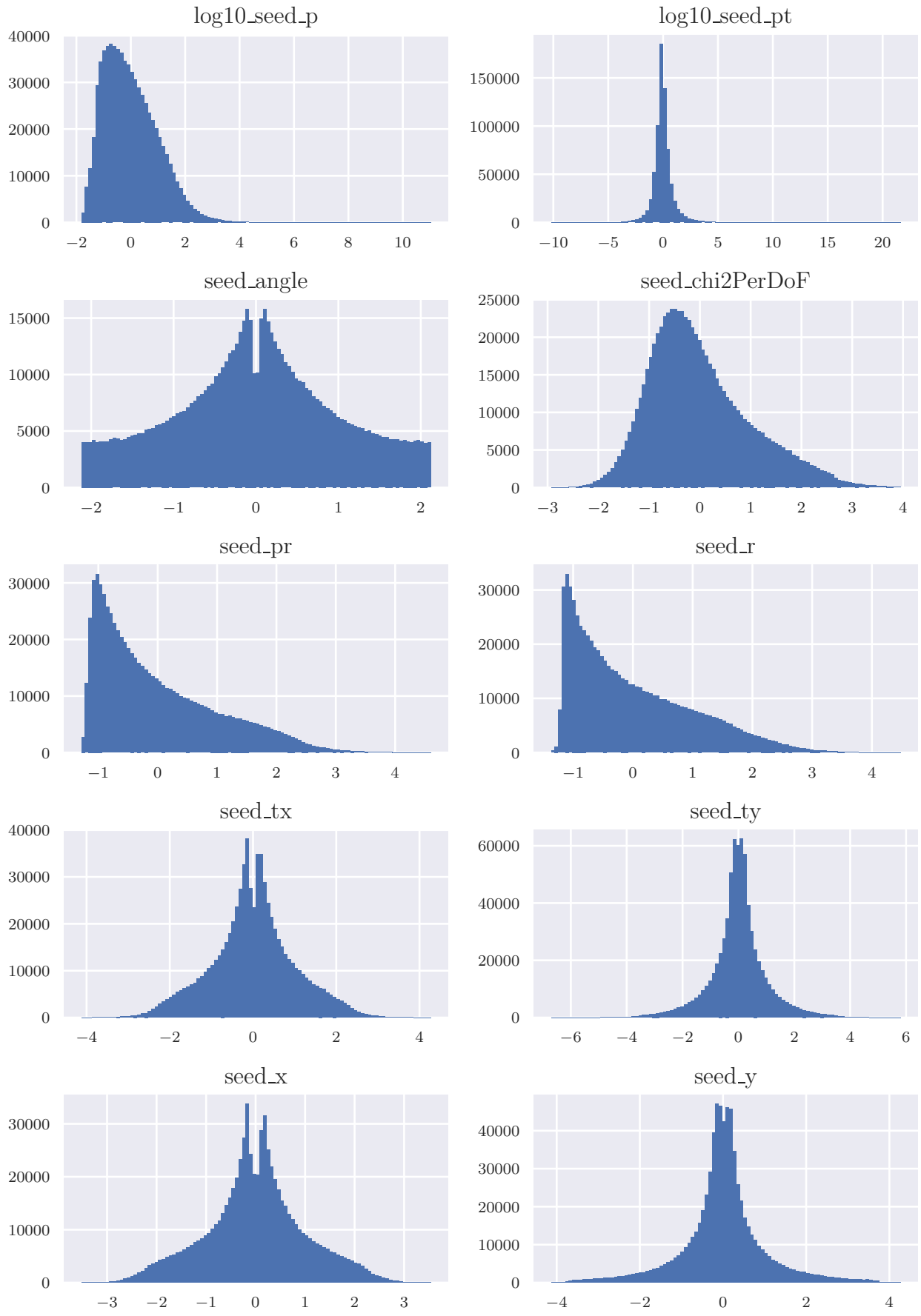

Figure 13: Histograms of training set features after the transformations and scaling.

4.2 Neural network architecture and training

The network model was implemented in Python using the Keras[31] library. The following hyperparameters were chosen experimentally and provided best training and validation scores.

The neural network model consists of 3 dense hidden layers with the Rectified Linear Units[32] (ReLu, see Figure 7) activations, each layer having 64 neurons. After every hidden dense layer the batch normalisation was used [24]. The output layer is set to be 1 neuron with sigmoid activation function. Summary of the model structure is presented in Table 5. The network was trained with the Adam optimiser using the default parameters[33] and batch size of 1000. The binary cross entropy was used as the loss function. The training took a total of 50 epochs. After each epoch, the loss and accuracy were measured on training and validation dataset (Figure 14). The final training and validation scores are shown in Table 6. No additional regularisation technique was used as the number of network parameters was small compared to the size of the training set and the model did not overfit the data.

Table 5: Summary of the neural network model architecture

Layer	Activation	Output size	Parameters
Input Layer	-	10	0
Dense 1	ReLu	64	704
Batch Normalisation 1	-	64	256
Dense 2	ReLu	64	4160
Batch Normalisation 2	-	64	256
Dense 3	ReLu	64	4160
Batch Normalisation 3	-	64	256
Dense 4	sigmoid	1	65
Total parameters			9 857
Trainable parameters			9 473
Nontrainable parameters			384

Table 6: Final performance on train and validation datasets. Accuracy measured with $th = 0.5$.

metric	training score	validation score
mean cross entropy loss	0.292	0.298
accuracy	0.881	0.878
ROC AUC	0.944	0.942

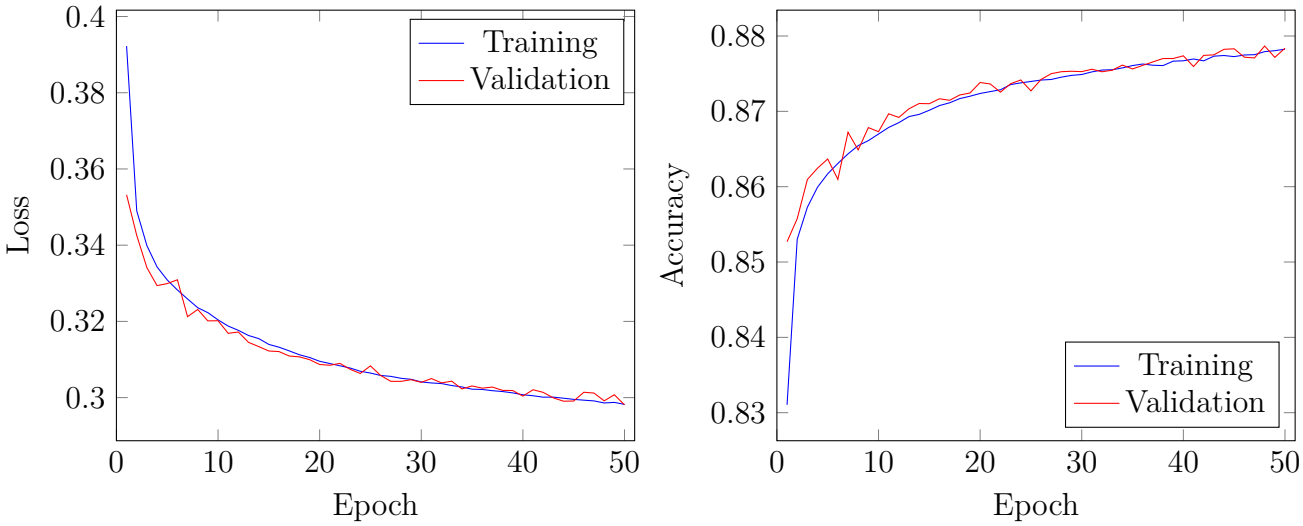

Figure 14: Binary cross entropy loss measured on training and validation datasets (left). Accuracy measured on training and validation sets (right).

4.3 Evaluation

After the model had been trained and the results on the validation set were similar the metrics were estimated on a held-out test set. Scores on this tests set are presented as the final performance results and can be compared with other models (Figure 15 and Table 7).

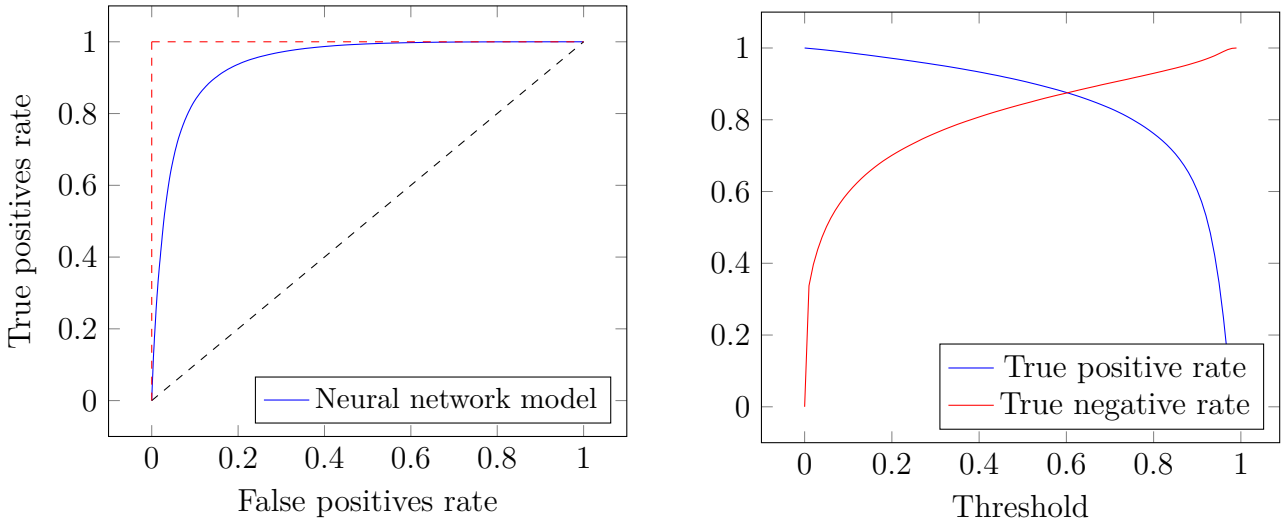

Figure 15: Receiver Operating Characteristic (ROC) curve (left). True positive rate and True negative rate as a function of prediction threshold (right).

Table 7: Performance of the new model on test dataset compared to BBDT classifier [5]. Accuracy measured with $th = 0.5$.

metric	test score	BBDT score
accuracy	0.88	0.73
ROC AUC	0.94	0.87

The classifier's imperfection forces a compromise between ghost rejection and signal preservation. Based in Figure 15 the decision threshold that keeps over 96% of the true seeds was be estimated to be $th_{(final)} = 0.245$. This corresponds to the true negative rate of 74% (Figure 16).

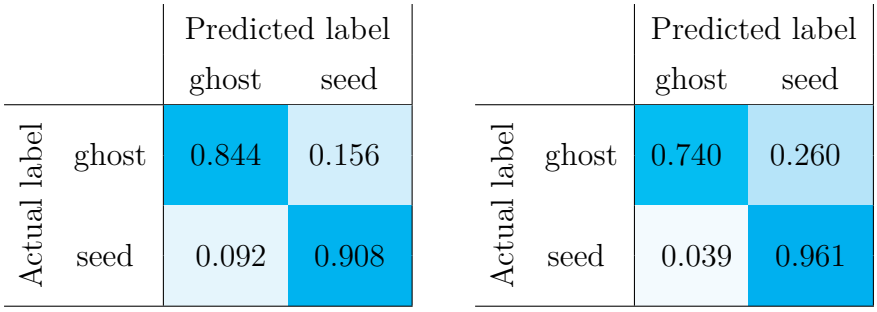

Figure 16: Confusion matrices for $th = 0.500$ (left) and $th_{(final)} = 0.245$ (right).

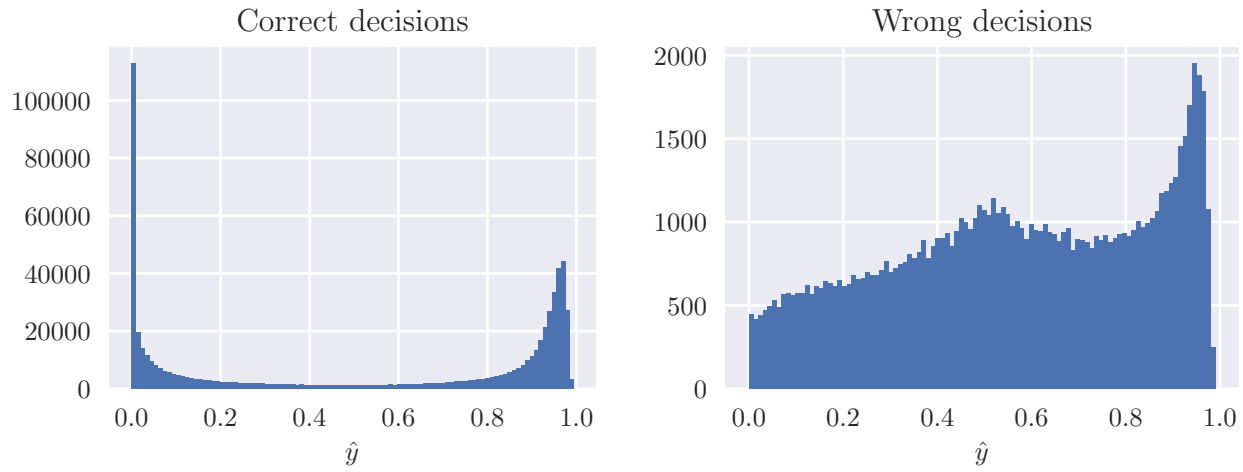

Figure 17: Histograms of model outputs on the test set. Correctly classified examples (left) and model mistakes (right). Visible large number of false positives with high model certainty.

The distribution of the model mistakes (Figure 17-right) gives clues for a source of model imperfect performance. The histogram shows a large number of examples which were incorrectly classified as true seeds with high certainty. Those examples have a dominant contribution to the loss function and prevent further improvements in performance during training. The second peak, located between the values of 0.4 and 0.6 was expected, as it corresponds to the range of highest model uncertainty.

4.4 Integration with LHCb software

To comply with the requirements of the LHCb software the model prediction has to be implemented in C++ and integrated into Brunel application. The direct porting of the model to Tensorflow [34] (a backend library of Keras with C++ API) was attempted. This approach failed due to the complicated structure of Tensorflow library and a large number of dependencies. Instead, a converter library, called *Lightweight Trained Neural Network*[35] (LWTNN) was used. LWTNN uses a minimum number of dependencies, all of which are already available in Gaudi. The necessary porting procedure is as follows:

1. Train model using Keras library in Python.
2. Save model architecture and weight to JSON files.
3. Convert saved files to LWTNN graph using tools provided by the library.
4. Load LWTNN graph in a C++ application and generate predictions.

The implementation extends *GaudiTool*[36] and *IPatMvaClassifier*[37] intefaces and provide basic functionalities required for generating predictions.

5 Physics performance

5.1 Figures of merit

To assess the effect of the new classifier on the performance of PatLongLivedTracking reconstruction efficiency, ghost rate (Tables 8 and 9) and CPU time (Table 10) have been measured using simulated data samples of $B^0 \to J/\psi K^0_s$ and $D^* \to D^0\pi$ decays.

The new classifier marginally reduces the reconstruction efficiency and requires more CPU time to generate predictions. However, the measured ghost rate is significantly lowered.

Table 8: PatLongLivedTracking reconstruction efficiency and ghost rate measured on simulated $B^0 \to J/\psi K^0_s$ decay using different seed classifiers.

parameter	new model	BBDT
ϵ_{rec}	74.1%	75.0%
$\epsilon_{rec,p>5GeV}$	80.5%	81.4%
ghost rate	23.2%	30.3%

Table 9: PatLongLivedTracking reconstruction efficiency measured on simulated $D^* \to D^0\pi$ decay using different seed classifiers.

parameter	new model	BBDT
ϵ_{rec}	73.1%	74.1%
$\epsilon_{rec,p>5GeV}$	79.9%	80.8%
ghost rate	24.2%	31.3%

Table 10: PatLongLivedTracking algorithm mean execution time per processed event using different seed classifiers.

parameter	new model [s]	BBDT [s]
CPU time	1.822	1.681

5.2 Impact on K_s^0 and Λ^0 mass measurement

Track reconstruction performance on the real collision data sample were studied by analysing the distributions of invariant mass of K_s^0 meson and Λ^0 baryon. The Gaussian function with linear background was fitted for both particles and the signal significance within 2σ of mean value was measured. Results are presented in Figure 18 and Table 11 for the K_s^0 and in Figure 19 and Table 12 for the Λ^0. Using the new classifier does not have a meaningful influence on the measured parameters.

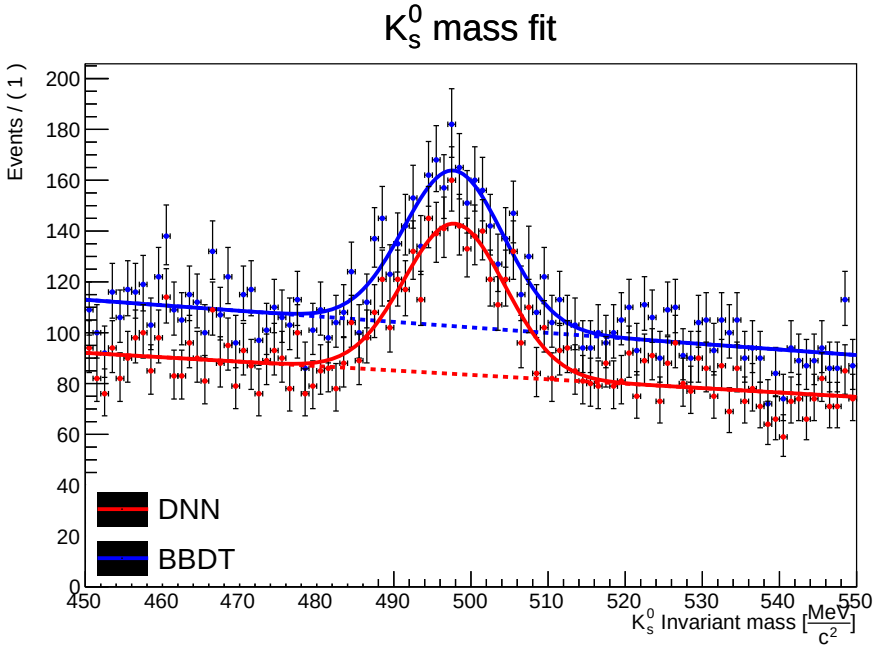

Figure 18: Invariant mass of K_s^0 meson measured using Bonsai Boosted Decision Trees classifier (BBDT) and neural network (DNN) with fitted Gaussian distribution (solid lines) and estimated linear background (dashed lines).

Table 11: Parameters of fitted Gaussian distribution with linear background for the K_s^0 invariant mass measured using Bonsai Boosted Decision Trees classifier (BBDT) and neural network (DNN). Mean and sigma expressed in $\frac{MeV}{c^2}$.

parameter	new model	BBDT
mean	497.86(50)	497.73(52)
sigma	6.55(50)	6.47(51)
signal events	15.68	15.48
background events	26.33	26.00
signal significance	2.42	2.40

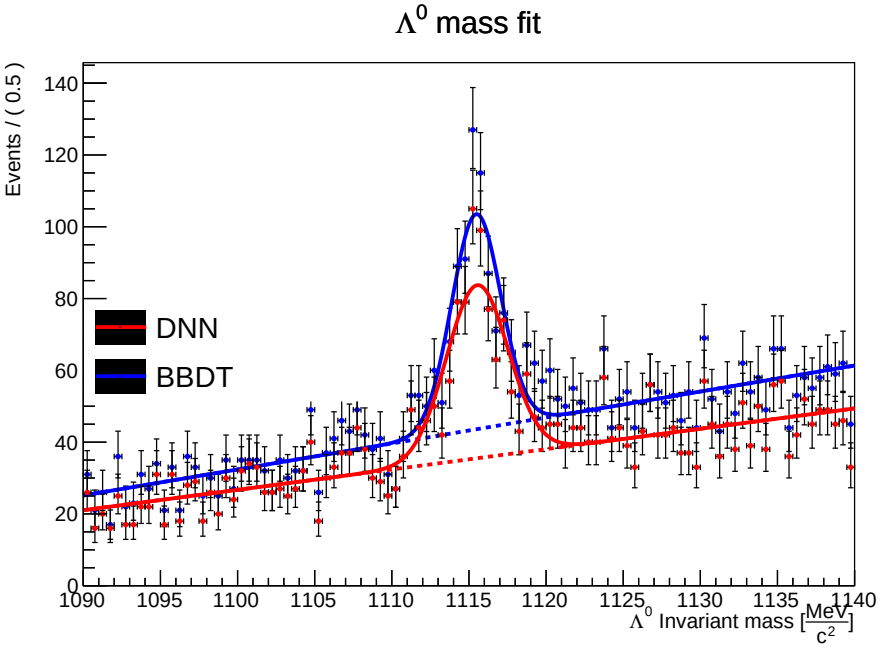

Figure 19: Invariant mass of Λ^0 baryon measured using Bonsai Boosted Decision Trees classifier (BBDT) and neural network (DNN) with fitted Gaussian distribution (solid lines) and estimated linear background (dashed lines).

Table 12: Parameters of fitted Gaussian distribution with linear background for the Λ^0 invariant mass measured using Bonsai Boosted Decision Trees classifier (BBDT) and neural network (DNN). Mean and sigma expressed in $\frac{MeV}{c^2}$.

parameter	new model	BBDT
mean	1115.52(15)	1115.47(15)
sigma	1.70(20)	1.59(21)
signal events	4.07	3.81
background events	6.87	6.42
signal significance	1.23	1.19

6 Conclusions

The goal of this thesis was to develop a T-seed classifier - a component of the long-lived particle track reconstruction algorithm in the LHCb experiment.

Presented methodology results in a neural network model, which can discriminate between simulated good and bad seeds with an accuracy of 88% and ROC AUC of 94%. The chosen decision threshold preserves 96% of the true seeds while rejecting 74% of ghosts. This corresponds to a decrease in reconstruction efficiency of 1.0% and track ghost rate by 7.1% as measured on simulated data. The PatLongLivedTracking average event processing time is increased by 0.14 s, as compared to a previous solution.

The improvements in performance were found to be too subtle to increase the K_s^0 and Λ^0 signal significance.

Introduction of LWTNN allows for separation of the model development stage, using convenient Python libraries, and deployment compatible with LHCb C++ libraries. The model export procedure has no negative impact on the classification performance.

The Author suggests three possible directions for further improvements.

The dominant prediction errors are false positives with high model confidence levels. Identifying the source of these mistakes will require generating a new dataset with additional T-seed debugging information.

The training was performed using purely simulated particles, which do not entirely match real detector conditions. Future work should incorporate the second source of training data, obtained from LHCb collisions.

The decision whether to reject a particular seed example based on just the features of one seed alone is another weak point of the current system. The classification task could potentially be replaced by a recommendation-like system, which decides which seeds are worth further processing. The decision would be made for all seeds at the same time, using full information available.

References

[1] A. A. Alves, Jr. et al., "The LHCb Detector at the LHC", JINST **3**, S08005 (2008).

[2] B. Adeva, et al., "Roadmap for selected key measurements of LHCb", (2009).

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", Nature **521**, 436 (2015).

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search", Nature **529**, 484 (2016).

[5] A. Davis, M. De Cian, A. M. Dendek, and T. Szumlak, *PatLongLivedTracking: A tracking algorithm for the reconstruction of the daughters of long-lived particles in LHCb*, tech. rep. LHCb-PUB-2017-001. CERN-LHCb-PUB-2017-001 (CERN, Geneva, Jan. 2017).

[6] G. Barrand, et al., "GAUDI - A software architecture and framework for building HEP data processing applications", Comput. Phys. Commun. **140**, 45–55 (2001).

[7] *LHCb layout*, `http://lhcb-geom.web.cern.ch/lhcb-geom/images/y-LHCb-reoptimized.pdf` (visited on 01/04/2018).

[8] *The GAUSS Project*, `http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/gauss/` (visited on 01/04/2018).

[9] *The BOOLE Project*, `http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/boole/` (visited on 01/04/2018).

[10] *The MOORE Project*, `http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/moore/` (visited on 01/04/2018).

[11] *The BRUNEL Project*, `http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/brunel/` (visited on 01/04/2018).

[12] *The DAVINCI Project*, `http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/davinci/` (visited on 01/04/2018).

[13] *The LHCb data flow - The LHCb Starterkit materials*, (visited on 01/01/2017).

[14] M. Vesterinen, A. Davis, and G. Krocker, *Downstream tracking for the LHCb upgrade*, tech. rep. LHCb-PUB-2014-007. CERN-LHCb-PUB-2014-007 (CERN, Geneva, Jan. 2014).

[15] S. Stahl, "Reconstruction of displaced tracks and measurement of Ks production rate in proton-proton collisions at $\sqrt{s} = 900$ GeV at the LHCb experiment", PhD thesis (Heidelberg University, June 2010).

[16] *Track types for the LHCb Run I and II*, https://twiki.cern.ch/twiki/pub/LHCb/ConferencePlots/trackTypes_upgrade.pdf (visited on 12/28/2017).

[17] T. Chen, and C. Guestrin, "Xgboost: A scalable tree boosting system", CoRR **abs/1603.02754** (2016).

[18] A. Rogozhnikov, *hep_ml 0.5.0 - Machine Learning for High Energy Physics*, https://arogozhnikov.github.io/hep_ml/ (visited on 01/04/2018).

[19] A. Frodesen, O. Skjeggestad, and H. Tfte, *Probability and statistics in particle physics*, Probability and Statistics in Particle Physics t. 2 (Universitetsforl., 1979).

[20] A. L. Samuel, "Some studies in machine learning using the game of checkers", IBM Journal of Research and Development **3**, 210–229 (1959).

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, http://www.deeplearningbook.org (MIT Press, 2016).

[22] P. Domingos, "A Few Useful Things to Know About Machine Learning", Commun. ACM **55**, 78–87 (2012).

[23] W. S. McCulloch, and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", The bulletin of mathematical biophysics **5**, 115–133 (1943).

[24] S. Ioffe, and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ArXiv e-prints (2015).

[25] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop", in Neural networks: tricks of the trade, edited by G. Orr, and M. K., (1998).

[26] S. Ruder, "An overview of gradient descent optimization algorithms", ArXiv e-prints (2016).

[27] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (O'Reilly Media, Mar. 2017).

[28] *Confusion matrix - scikit-learn 0.19.1 documentation*, http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py (visited on 12/17/2017).

[29] *Receiver Operating Characteristic - scikit-learn 0.19.1 documentation*, (2017) http://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics (visited on 12/17/2017).

[30] A. Karpathy, *CS231n: Convolutional Neural Networks for Visual Recognition - Lecture notes*, http://cs231n.github.io/neural-networks-2/ (visited on 01/04/2017).

[31] F. Chollet, et al., *Keras*, https://github.com/fchollet/keras, 2015.

[32]V. Nair, and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in Proceedings of the 27th international conference on international conference on machine learning, ICML'10 (2010), pp. 807–814.

[33]D. P. Kingma, and J. Ba, "Adam: A Method for Stochastic Optimization", ArXiv e-prints (2014).

[34] Abadi, M, et al., *TensorFlow: large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[35]*Lightweight trained neural network*, (2017) `https://github.com/lwtnn/lwtnn` (visited on 12/17/2017).

[36]V. Belyaev, and C. Jones, *GaudiTool Class Reference*, `http://lhcb-doxygen.web.cern.ch/lhcb-doxygen/davinci/latest/d5/d28/_gaudi_tool_8h.html` (visited on 01/04/2018).

[37]A. Dendek, *IPatMvaClassifier Class Reference*, `http://lhcb-doxygen.web.cern.ch/lhcb-doxygen/davinci/latest/d3/d66/struct_i_pat_mva_classifier.html` (visited on 01/04/2018).