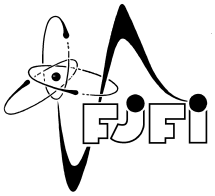
Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering



Supervised and Unsupervised Learning for Heavy Ion Physics

Supervizované a nesupervizované učení pro fyziku těžkých iontů

Master's Thesis

Author: **Bc. Kateřina Hladká**

Supervisor: **Ing. Václav Kůs, Ph.D.**

Consultant: **prof. Dr. Boris Tomášik,**
Iurii Karpenko, Ph.D.

Academic year: 2020/2021

Katedra: matematiky Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Kateřina Hladká
Studijní program:	Aplikace přírodních věd
Studijní obor:	Aplikované matematicko-stochastické metody
Název práce (česky):	Supervizované a nesupervizované učení pro fyziku těžkých iontů
Název práce (anglicky):	Supervised and Unsupervised Learning for Heavy Ion Physics

Pokyny pro vypracování:

1) Nastudujte problematiku QCD hmoty se zaměřením na povahu QCD fázových přechodů v relativistických srážkách těžkých jader. Seznamte se se simulovaným datovým souborem odpovídajícím různým typům těchto přechodů.
2) Zabývejte se problematikou reziduálních a konvolučních neuronových sítí. Vytvořte optimalizovaný klasifikátor vzhledem k fyzikálně vhodné metrice a proveďte klasifikaci simulovaného datového souboru dle typu fázového přechodu. Pro vybrané reprezentanty daných tříd zkoumejte vývoj tzv. feature maps v jednotlivých vrstvách. Výsledky diskutujte s fyzikální komunitou.
3) Seznamte se s novým typem dat pro simulace rozpadu mezonu D0, respektive jeho antimezonu (potenciálně poskytnutých KF při FJFI). Na základě poznatků z bakalářské práce a výzkumného úkolu navrhněte optimalizovaný klasifikátor a porovnejte jeho kvalitu s předchozími dosaženými výsledky.
4) Seznamte se s principy nesupervizovaného učení (např. self organizing maps, k-means...). Navrhněte modifikaci vybrané metody tak, aby bylo možné provést shlukovou analýzu dle zvoleného informačně-divergenčního kritéria pro HEP data z rozpadů těžkých jader.

Doporučená literatura:

1) C. C. Aggarwal, Neural Networks and Deep Learning. Springer International Publishing, 2018.

2) C. M. Bishop, Pattern Recognition and Machine Learning. Springer-Verlag New York, 2006.

3) K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition. In '2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', IEEE Xplore, 2016, 770-778.

4) L. Pardo, Statistical inference based on divergence measures. Chapman & Hall/CRC, 2006.

5) Y. Du, K. Zhou, J. Steinheimer, L. Pang, A. Motornenko, H. Zong, X. Wang, H. Stöcker, Identifying the nature of the QCD transition in relativistic collision of heavy nuclei with deep learning. The European Physical Journal C 80, 2020, 516:1-17.

Jméno a pracoviště vedoucího diplomové práce:

Ing. Václav Kůs, Ph.D.
Katedra matematiky, FJFI ČVUT Praha, Trojanova 13, 120 00 Praha 2

Jméno a pracoviště konzultanta:

prof. Dr. Boris Tomášik, Ph.D., Iurii Karpenko, Ph.D.
Katedra fyziky, FJFI ČVUT Praha, Břehová 7, 115 19 Praha 1

Datum zadání diplomové práce: 31.10.2020

Datum odevzdání diplomové práce: 3.5.2021

Doba platnosti zadání je dva roky od data zadání.

Acknowledgment:
I would like to thank my supervisor, Ing. Václav Kůs, Ph.D., for his expert guidance and patience throughout the past 3 years. I'm extremely grateful to my consultants, prof. Dr. Boris Tomášik and Iurii Karpenko, Ph.D., for the support, many expert insights and helpful advices during the project. I gratefully acknowledge the effort of Ing. Jakub Cimerman and Ing. Lukáš Kramárik when providing the data used for the purposes of this project. I also very much appreciate STAR collaboration's willingnes to provide HIJING simulation used in this thesis. I must also thank to doc. Mgr. Jaroslav Bielčík, Ph.D. for continuous support of my work during past 3 years. Last but very special thanks to my family and partner.

Author's declaration:
I declare that this master thesis is entirely my own work and I have listed all the used sources in the bibliography.

Prague, May 3, 2021 Kateřina Hladká

Název práce:

Supervizované a nesupervizované učení pro fyziku těžkých iontů

Autor: Kateřina Hladká

Obor: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: Ing. Václav Kůs, Ph.D., Katedra matematiky, FJFI ČVUT Praha

Konzultanti: prof. Dr. Boris Tomášik, Ph.D. a Iurii Karpenko, Ph.D., Katedra fyziky, FJFI ČVUT Praha

Abstrakt: Potřeba řešení komplexních problémů binární klasifikace ve fyzice těžkých iontů vyústila v častější užívání supervizovaných i nesupervizovaných algoritmů strojového učení. Spolu s vhodným předzpracováním dat a optimalizací hyperparametrů tak tvoří silný nástroj pro řešení takovýchto problémů. Tato práce porovnává několik přístupů strojového učení a efektu více typů předzpracování dat pro dva problémy fyziky těžkých iontů: identifikaci produktů rozpadu D^0 a identifikaci povahy QCD fázového přechodu. Metody použité v této práci zahrnují náhodný les, hluboké neuronové sítě a konvoluční neuronové sítě s i bez reziduálních bloků.

Klíčová slova: binární klasifikace, D^0 rozpad, optimalizace, QCD fázové přechody, strojové učení

Title:

Supervised and Unsupervised Learning for Heavy Ion Physics

Author: Kateřina Hladká

Abstract: The need of solving complex binary classification problems in the heavy ion physics has resulted in the usage of both supervised and unsupervised machine learning algorithms. Together with appropriate data preprocessing and hyper-parameters optimization they form a strong tool for addressing such problems. This study compares multiple machine learning approaches and data pre-processing dependencies for two heavy ion physics problems: D^0 decay classification and identification of the nature of the QCD phase transition. The methods used in this thesis include random forest, deep neural networks and convolutional neural networks with and without residual blocks.

Key words: binary classification, D^0 decay, machine learning, optimization, QCD phase transition

Contents

Introduction

Nowdays, complex data science challenges of heavy-ion physics are handled by introducing advanced techniques of machine learning. This study aims to explore two binary classification problems, identification of D^0 decay products and identification of the QCD phase transition nature, and their possible solution using several machine learning approaches. Furthermore, appropriate data pre-processing and trained model's performance dependency on given pre-processing type was explored as well.

The thesis is divided into four parts. Chapters 1–3 describe theoretical background. Specifically, Chapter 1 introduces challenges of D^0 decay analysis and QCD phase transitions as well as the data that were used for machine learning models training. Chapter 2 describes supervised machine learning algorithms that were used to address challenges defined in Chapter 1, such as random forests, deep neural networks, convolutional neural networks and residual neural networks. In addition, techniques of optimization are introduced in Chapter 2. Chapter 3 proposes unsupervised machine learning algorithms suitable for usage in heavy-ion physics, e.g., self-organizing maps, and completes the theoretical part of the thesis. Finally, Chapter 4, presents multiple data science pipelines including the statistical tests of hypotheses for suitable representation of binary classes, data pre-processing, optimizing machine learning models and their evaluation and comparison over test sets.

Chapter 1

Data science challenges of heavy-ion physics

1.1 D^0 decay

Undoubtedly one of the most interesting state of matter studied by the modern physics is the quark-gluon plasma (QGP). Fractions of a second after the Big Bang, all matter existed in such form due to high temperature and pressure. Today, heavy-ion collisions are used to recreate similar conditions in order to produce small amount of QGP. So far, current detectors are not able to detect quark-gluon plasma itself, since is exists only for an extremely short period of time. However, studying final-state particles coming from the QGP or decay products of short-lived particles is possible and reverse process reconstruction may bring new insights to the research field.

One of the experiments taking place in Brookhaven National Laboratory is the STAR experiment named after the same named detector working at the Relativistic Heavy Ion Collider. Among other things, its purpose is to study of heavy-ion collision products, one of them being meson D^0. It is not possible to detect meson D^0 itself. However, products of its decay, pions π and kaons K, can be detected and measured in terms of physical properties listed in Table 1.1.

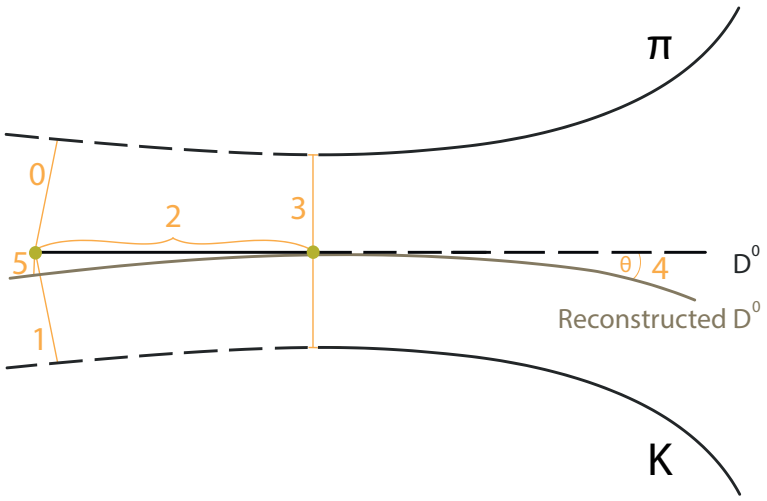


Figure 1.1: D^0 decay scheme.

A challenging classification problem is inevitably triggered: which pairs of pions π and kaons K are truly D^0 decay products (signal) and which detected pairs are just combinatorial background? In [1] and [2] we introduced a solution based on random forest, AdaBoost and deep learning approach built over PYTHIA signal simulation and measured data provided by STAR collaboration. Each data point

id	shortcut	description	pre-cut criterion
0	DCA_{π}	distance of the closest approach of π to primary vertex [cm]	$0.002 < DCA_{\pi} < 0.2$
1	DCA_{K}	distance of the closest approach of K to primary vertex [cm]	$0.002 < DCA_{K} < 0.2$
2	l_{decay}	D^0 decay length [cm]	$0.0005 < l_{decay} < 0.2$
3	$DCA_{\pi,K}$	distance of the closest approach of K and π [cm]	$DCA_{\pi,K} < 0.02$
4	$\cos\theta$	cosine of the pointing angle θ between reconstructed D^0 momentum and decay length vector	$\cos\theta > 0.7$
5	DCA_{D^0}	distance of the closest approach of the reconstructed D^0 and primary vertex [cm]	$DCA_{D^0} < 0.05$
6	$\cos\theta^*$	cosine of the pointing angle between reconstructed D^0 momentum and K momentum	
7	${D^0}_{mass}$	D^0 invariant mass [GeV/c^2]	
8	p_T	transverse momentum of D^0 [GeV/c]	

Table 1.1: Features of K and π with pre-cut criteria

in PYTHIA simulated data set and measured data set was described by 9 features as listed in Table 1.1. Unlike the simulation, measured data were subdivided into two disjoint subsets, unlike-sign set and like-sign set. The division was based on the D^0 charge nature which conditioned the required combination of its decay products' charges. Since D^0 meson decay $D^0 \rightarrow K^- + \pi^+$ was used, unlike-sign K π pairs are expected to contain D^0 signal together with some combinatorial background. In order to enhance size of the sample, $\overline{D^0} \rightarrow K^+ + \pi^-$ was also taken as signal. Naturally, not all unlike-sign pairs are signal pairs. However, all like-sign pairs are background pairs as described in Fig. 1.2. Assuming that the like-sign background is homogeneous with unlike-sign background in terms of joint probability distribution of features 0–8 from Table 1.1, it may be used as overall background representation. By fusion with PYTHIA simulation learning set for training the classification model is created. This was indeed performed in [1] and [2]. Learning set was divided into three subsets based on p_T value of the data points, pre-cut criteria listed in Table 1.1 were applied to all individual samples. Three models were trained over matching data subset and later applied to similarly pre-processed unlike-sign subset of the measured data to filter out the signal candidates. The quality of classification was studied not only using traditional binary classification metrics, but also by defining the concept of generalized significance. Different machine-learning algorithms were optimized and tested within proposed pipeline. Later, comparison to the approach of boosted decision trees, a concept widely used in high energy data community, was performed.

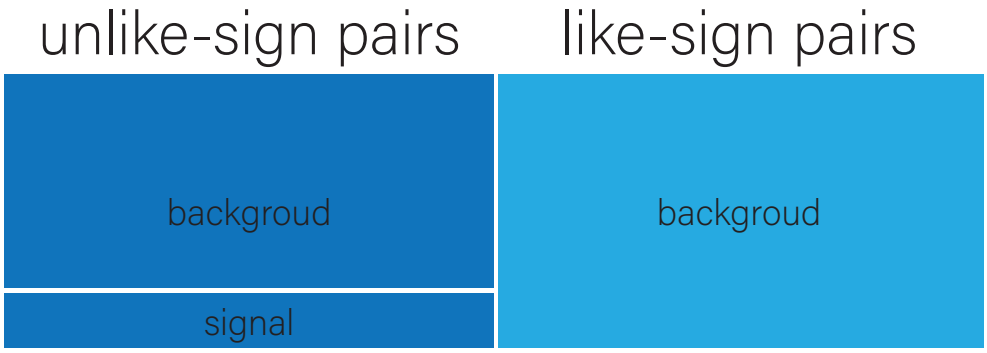


Figure 1.2: Schema of measured data structure.

For purposes of this thesis, new simulation (HIJING) was provided instead of PYTHIA simulation. Unlike PYTHIA, HIJING contains simulation of unlike-sign and like-sign background as well. The representation of the signal is of better quality in terms of reality representation. The new simulation structure triggered multiple data science and/or statistical questions:

- Is the simulated unlike-sign background statistically homogeneous with the like-sign background? If not, how the signal class should be represented during the training?
- Is the simulated like-sign background statistically homogeneous with the like-sign background of measured data?
- What are the statistical differences between PYTHIA and HIJING?
- If the HIJING signal and background representation is "similar enough" to PYTHIA simulation and like-sign background of measured data, how successful will be the model used in [2] when applied to HIJING simulation?
- Since the model trained over HIJING should be validated over HIJING as well (and not over unlike-sign set of measured data), what is the most suitable metrics to use for evaluation instead of generalized significance as proposed in [2]?
- What is the quality of chosen Python 3 classification model compared to the widely used boosted decision tree model implemented using The Toolkit for Multivariate Data Analysis (TMVA) in ROOT programming language?

Questions stated above and the development of new classification models are to be described by this thesis.

1.2 QCD transitions

When studying QGP attributes, one could ask the question what conditions have to be met to create such state of matter. According to the theory of quantum chromodynamics (QCD), matter in a form of hadron gas makes smooth crossover transition to QGP at the high temperature T ~140 – 180 MeV (2×10^{12} K) while the net baryon density is low. Theoretical models assume that another type of phase transition (first order) should be expected when the net baryon density is high. However, this has yet to be confirmed both theoretically and experimentally. Inconspicuously, this leads us to another data science challenge. How the type of phase transition can be detected when QGP is formed? In [3], prediction model built over pion spectra retrieved from simulated high energy nucleus-nucleus collisions was introduced. Naturally, this has opened the question, which parameters of the simulation were significant for the phase transition prediction (including the energy of simulated collision itself). This thesis aims to explore performance dependence of phase-transition model inspired by [3] on simulation parameters such as collision energy or centrality class. Centrality class is defined as a measure of how precisely the nuclei hit each other tip-on-tip, with 0 % meaning exact overlap of the two nuclei and 100 % just grazing each other.

Data description

Data were simulated using a hybrid hydrodynamic model with viscous relativistic hydrodynamics [4]. After the Au+Au nucleus-nucles collision, detector is able to detect some of its products, for instance, pions π^+, π^0 and π^-. Each of the pions may be described by its detected energy E and momentum $\overrightarrow{p} = (p_x, p_y, p_z)$. Using this, other variables are derived such as azimuthal angle Φ

$$\Phi = \arctan \frac{p_y}{p_x}, \tag{1.1}$$

transverse momentum p_{T}

$$p_{\mathrm{T}} = \sqrt{p_x^2 + p_y^2}, \tag{1.2}$$

and rapidity y

$$y = \frac{1}{2} \ln \frac{E + p_z}{E - p_z}. \tag{1.3}$$

In accordance with [3], only pions meeting the condition $p_{\mathrm{T}} \leq 2$ GeV/c, $|y| \leq 1$ and $\Phi \in [0, 2\pi]$ are later used to produce an input for the binary classification model. For given set of simulation parameters and equation of state (EoS), .root file containing set of events is available. For each event, pions meeting the requirements on p_{T}, y and Φ value are filtered out and discrete representation of (p_{T}, Φ) joint distribution is produced (raw histogram). Later, in chapter 4, classifier's robustness towards event rotation within transverse plane (p_x, p_y) will be tested due to presence of such effect in real collisions. For such scenario, random angle $\alpha \sim U(-\pi, \pi)$ is generated for each event and p_x, p_y of each pion within given event is transformed in original .root data set. Transverse momentum p_{T}, rapidity y and azimuthal angle Φ are calculated. Based on their values, pions are filtered to produce discrete representation of (p_{T}, Φ) joint distribution (raw rotated histogram). Additional pre-processing types are to be described in chapter 4.

Chapter 2

Supervised machine learning

2.1 Binary classification

Definition 2.1.1. *Let* $\mathbf{x_i} = (x_{i1}, ..., x_{ir})$*, where* $x_{ij} \in \mathbb{R}$*,* $i \in \{1, ..., n\}$*,* $j \in \{1, ..., r\}$ *and* $r, n \in \mathbb{N}$ *be observation of random variable* $X = (X_1, ..., X_r)$ *from sample space* Ω*. Let* C_0*,* C_1 *be two disjoint classes, such as for every* $i \in \{1, ..., n\}$

$$I_{C_0}(\mathbf{x_i}) = 1 \qquad \wedge \qquad I_{C_1}(\mathbf{x_i}) = 0 \qquad \Leftrightarrow \qquad \mathbf{x_i} \in C_0 \tag{2.1}$$

or

$$I_{C_1}(\mathbf{x_i}) = 1 \qquad \wedge \qquad I_{C_0}(\mathbf{x_i}) = 0 \qquad \Leftrightarrow \qquad \mathbf{x_i} \in C_1, \tag{2.2}$$

where I_{C_0} *and* I_{C_1} *are indicators of classes* C_0*,* C_1*. Let* $\mathbf{y} = (y_1, ..., y_n)$ *be observations of random variable Y, such as for every* $i \in \{1, ..., n\}$*,* $y_i = I_{C_1}(\mathbf{x_i})$*. Then, learning set L is defined as set of tuples* $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\} = (\mathbf{X}, \mathbf{y})$*.*

As indicated by definition 2.1.1, the goal is to find the classifier $\mathscr{C}$ estimating probability $P(Y \mid X = \mathbf{x}_i)$, which assigns class C_0 or C_1 to the sample $\mathbf{x_i}$. Let's denote prior a probability of event $I_{C_1}(\mathbf{x_i}) = 1$ as p_1. Similarly, prior probability of event $I_{C_1}(\mathbf{x_i}) = 0$ will be denoted p_0. Using Bayes rule

$$P(Y = 0 \mid \mathbf{x_i}) = \frac{p_0 P(X = \mathbf{x_i} \mid Y = 0)}{P(X = \mathbf{x_i})} \tag{2.3}$$

and

$$P(Y = 1 \mid \mathbf{x_i}) = \frac{p_1 P(X = \mathbf{x_i} \mid Y = 1)}{P(X = \mathbf{x_i})}. \tag{2.4}$$

This enables us to redefine conditions for $I_{C_0}(\mathbf{x_i})$ and $I_{C_1}(\mathbf{x_i})$ as

$$P(Y = 0 \mid \mathbf{x_i}) > P(Y = 1 \mid \mathbf{x_i}) \qquad \Leftrightarrow \qquad \mathbf{x_i} \in C_0 \tag{2.5}$$

and

$$P(Y = 1 \mid \mathbf{x_i}) > P(Y = 0 \mid \mathbf{x_i}) \qquad \Leftrightarrow \qquad \mathbf{x_i} \in C_1. \tag{2.6}$$

By this moment, classifier $\mathscr{C}$ itself may be defined.

Definition 2.1.2. *Let $l(x)$ be a likelihood ratio*

$$l(\mathbf{x}_i) = \frac{P(\mathbf{x}_i|Y=1)}{P(\mathbf{x}_i|Y=0)}. \tag{2.7}$$

Binary classifier is then defined as $\mathscr{C}: \Omega \to \{0,1\}$, such as

$$\mathscr{C}(\mathbf{x}_i) = \begin{cases} 1, & \text{if } l(\mathbf{x}_i) > \dfrac{p_0}{p_1}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

Typically, p_0 and p_1 are unknown parameters when dealing with binary classification problems. Therefore, ratio $\frac{p_0}{p_1}$ is replaced by threshold δ which may be optimized with respect to maximization of given binary classification metrics.

Estimating quality of the binary classifier $\mathscr{C}$

Definition 2.1.3. *Let be $\mathscr{C}$ a binary classifier $\mathscr{C}: \Omega \to \{0,1\}$. Let l be a loss function l: $\{0,1\}\times\{0,1\} \to \mathbb{R}$. Then, general error of the classifier $\mathscr{C}$ is defined as*

$$G(\mathscr{C}) = \mathbb{E}_{X_1,\ldots,X_r,Y}\left[l\left(Y, \mathscr{C}(X)\right)\right]. \tag{2.9}$$

Direct evaluation of general error is not possible for obvious reasons. The fact that learning set L contains finite number of tuples, creates also the need of forming at least two disjoint subsets. One would be used to find the mapping from X to Y (training stage) and the second subset would be used to evaluate the quality of $\mathscr{C}$. Such approach would prevent us of finding sufficient mapping over limited number data samples (over-training the classifier $\mathscr{C}$) while at the same time being unable to perform sufficient predictions over additional samples from sample space Ω.

Definition 2.1.4. *Let $n, p \in \mathbb{N}$ such as $1 << p << n$. Let $L = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$ be a learning set. Train set L_{train} is then defined as random subset of p tuples $(\mathbf{x_i}, y_i)$, where $i \in \{1, ..., n\}$. Next, test set L_{test} is defined as $L \setminus L_{train}$.*

Definition 2.1.5. *Let $x_i \in C_0$ for $i \in \hat{n}$. Prediction $\mathscr{C}(\mathbf{x}_i) = 0$ is called true negative. Prediction $\mathscr{C}(\mathbf{x}_i) = 1$ is called false positive.*

Definition 2.1.6. *Let $x_i \in C_1$ for $i \in \hat{n}$. Prediction $\mathscr{C}(\mathbf{x}_i) = 1$ is called true positive. Prediction $\mathscr{C}(\mathbf{x}_i) = 0$ is called false negative.*

Definition 2.1.7. *Let L^* be a subset of learning set L. Number of true positive predictions made by classifier $\mathscr{C}$ over set L^* is denoted by TP. Number of true negative predictions made by classifier $\mathscr{C}$ over set L^* is denoted by TN. Number of false positive predictions made by classifier $\mathscr{C}$ over set L^* is denoted by FP. Number of false negative predictions made by classifier $\mathscr{C}$ over set L^* is denoted by FN.*

Definition 2.1.8. *Let L^* be a subset of learning set L. Confusion matrix over for subset L^* is defined as*

$$\begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}.$$

Defining TP, FP, FN and FP will alow us to describe classification quality of the classifier $\mathscr{C}$ from multiple point of views.

Definition 2.1.9. *Precision (positive predicted value, signal purity) is defined as*

$$P = PPV = \frac{TP}{TP + FP}. \tag{2.10}$$

Recall (true positive rate, signal efficiency) is defined as

$$R = TPR = \frac{TP}{TP + FN} . \tag{2.11}$$

Negative predicted value (background efficiency) is defined as

$$NPV = \frac{TN}{TN + FN} . \tag{2.12}$$

True negative rate is defined as

$$TNR = \frac{TN}{TN + FP} . \tag{2.13}$$

False positive rate is defined as

$$FPR = \frac{FP}{FP + TN} . \tag{2.14}$$

Accuracy is defined as

$$ACC = \frac{TP + TN}{TP + TN + FN + FP} . \tag{2.15}$$

The definition above states the most common binary classification metrics. In some cases, especially when $\mathscr{C}$ contains hyper-parameters to be optimized, more complex decomposition of learning set L is required. Binary classification metrics are then evaluated on such subsets.

Definition 2.1.10. *Let $\{L_1, ..., L_k\}$ be disjoint subsets of learning set L for $k \in \mathbb{N}$. Let $\{L_{1\ train}, ..., L_{k\ train}\}$ be set of training sets, such as $L_{i\ train} = L \setminus L_i$ for every $i = 1, ..., k$. By k-fold cross validation we understand training the classifier $\mathscr{C}$ over $L_{i\ train}$ and evaluating chosen binary classification metrics over L_i. The overall performance of the classifier $\mathscr{C}$ is then derived as a statistics of k performances over $\{L_1, ..., L_k\}$.*

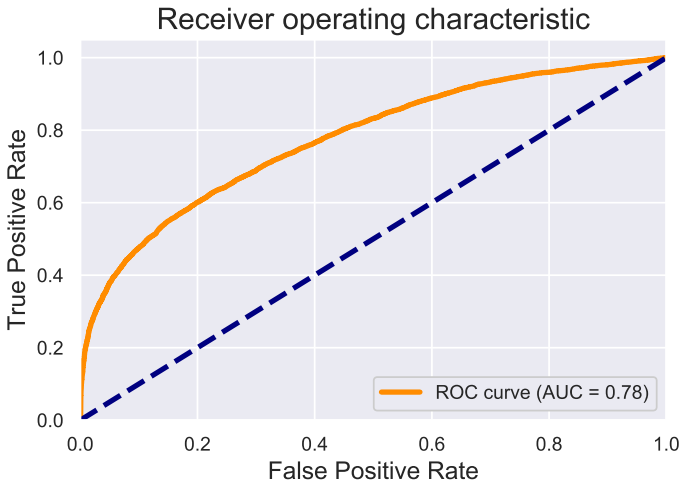


Figure 2.1: Example of ROC.

All the definitions introduced in this section are indirectly dependent on the threshold δ (or its optimized value with respect to given binary classification metrics). Therefore, it would be useful to map trade-off between some binary classification metrics values and threshold values. Such relationship may be tracked using receiver operating characteristics (ROC). ROC provides visualization of TPR and FPR dependency on different threshold values $\delta \in [0, 1]$ used during the classification process. The example of ROC is displayed in Fig. 2.1. The more ROC curve tends to both coordinates [0,1], the better is the classification. To evaluate the closeness of approach, the area under curve (AUC) is estimated.

2.2 Ensembles of trees

Contrary to the trend of investing many resources to the deep learning research, empirically, binary classification over data consisting of continuous features is in the most cases still better performed by robust, easily interpretable and fast-to-train ensembles of trees. In [1] we have introduced the most common algorithms such as random forests and shallow decision trees linked with AdaBoost algorithm. To compare two approaches for D^0 binary classification different enough with respect to boosted decision trees, deep neural networks and random forests were selected. In this chapter, theoretical summary of random forest algorithm and its optimization will be described.

Decision trees

Binary decision tree can be represented as oriented and rooted tree graph. For binary decision trees, every two nodes may be connected with exactly one path. This path (oriented vertex) determines parent node - child node relationship. The node with no parent is defined as the root node. A node with no children is defined as the leaf node. But how the idea of decision tree graph may be used in binary classification problems? We will demonstrate the classification approach on shallow tree with depth equal to one. In general, classification for arbitrarily deep tree can be done by applying the process of growing child nodes from root node to child nodes themselves. Let $L = (\mathbf{X}, y)$ be a learning set. Let n_0 denote the root and parent node of n_1 and n_2. Symbolically, let n_0 contains samples $\mathbf{X}$, which can be either assigned to class C_0 or C_1. Performing binary classification means to subdivide $\mathbf{X}$ into two disjoint subsets, one which will be contained in n_1, second which will be contained in n_2. The rule to perform such partition is provided by appropriate subset purity metrics introduced below.

Definition 2.2.1. *Impurity function of binary partition is defined as mapping $\phi(p^0, p^1)$: $[0,1] \times [0,1] \to \mathbb{R}$ such as*

- *ϕ is concave function*
- $p^0 + p^1 = 1$
- $\underset{p^0,p^1}{argmax}\ \phi = \left(\frac{1}{2}, \frac{1}{2}\right)$
- $\underset{p^0,p^1}{argmin}\ \phi = \{(1,0),(0,1)\}$
- *ϕ is symmetrical with respect to p^0, p^1.*

Definition 2.2.2. *Let $\widehat{\mathbf{X}} \subseteq \mathbf{X}$. Let n_j be a node of decision tree containing $\widehat{\mathbf{X}}$. Let $\widehat{\mathbf{X}}_{C_0} \subseteq \widehat{\mathbf{X}}$ be a subset of all samples form C_0. Let $\widehat{\mathbf{X}}_{C_1} \subseteq \widehat{\mathbf{X}}$ be a subset of all samples form C_1. Impurity measure $i(n_j)$ of node n_j is defined as $i(n_j) = \phi\left(p^0_{n_j}, p^1_{n_j}\right)$, where*

$$p^0_{n_j} = \frac{|\widehat{\mathbf{X}}_{C_0}|}{|\widehat{\mathbf{X}}|} \qquad \text{and} \qquad p^1_{n_j} = \frac{|\widehat{\mathbf{X}}_{C_1}|}{|\widehat{\mathbf{X}}|}.$$

Definition 2.2.3. *Let $\mathbf{B}$ be a family of binary partitions available to use for partitioning of subset $\widehat{\mathbf{X}} \subseteq \mathbf{X}$ contained in the node n_j. Let $b \in \mathbf{B}$ divide n_j into child n_{jL} and child n_{jR} such that $\widehat{\mathbf{X}}_L \subseteq \widehat{\mathbf{X}}$ is contained in n_{jL} and $\widehat{\mathbf{X}}_R \subseteq \widehat{\mathbf{X}}$ is contained in n_{jR}. Decrese of impurity caused by binary partitioning b is defined as*

$$\Delta i(b, n_j) = i(n_j) - \frac{|\widehat{\mathbf{X}_L}|}{|\widehat{\mathbf{X}}|} i(n_{jL}) - \frac{|\widehat{\mathbf{X}_R}|}{|\widehat{\mathbf{X}}|} i(n_{jR}). \tag{2.16}$$

Definition 2.2.4. *Gini impurity measure of node n_j is defined as*

$$i_G(n_j) = \sum_{k=0}^{1} p_{n_j}^k (1 - p_{n_j}^k). \tag{2.17}$$

Entropy impurity measure of node n_j is defined by

$$i_H(n_j) = -\sum_{k=0}^{1} p_{n_j}^k \log p_{n_j}^k. \tag{2.18}$$

Definition 2.2.5. *Let $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\}$ be a learning set L. Let $\mathbf{x_i} = (x_{i1}, ..., x_{ir})$, where $x_{ij} \in \mathbb{R}$, $i \in \{1, ..., n\}$, $j \in \{1, ..., r\}$ and $r, n \in \mathbb{N}$ be observation of random variable $X = (X_1, ..., X_r)$. Let $\widehat{\mathbf{X}} \subseteq \mathbf{X}$. Let $m \in \hat{r}$ and $\varepsilon \in \mathbb{R}$. Binary partitioning $b_\varepsilon^{X_m} \in \mathbf{B}$ of $\widehat{\mathbf{X}}$ by X_m is defined as creating two disjoint subsets $\widehat{\mathbf{X}}_L$ and $\widehat{\mathbf{X}}_R$ such that*

- $\widehat{\mathbf{X}}_L \cup \widehat{\mathbf{X}}_R = \widehat{\mathbf{X}}$,
- $\widehat{\mathbf{X}}_L \neq \emptyset \neq \widehat{\mathbf{X}}_R$,
- *for every* $\mathbf{x_i} \in \widehat{\mathbf{X}}_L$, $x_{im} < \varepsilon$,
- *for every* $\mathbf{x_i} \in \widehat{\mathbf{X}}_R$, $x_{im} \geq \varepsilon$.

Optimal binary partitioning $b_{\varepsilon^*}^{X_m} \in \mathbf{B}$ of $\widehat{\mathbf{X}}$ by X_m is found as

$$b_{\varepsilon^*}^{X_m} = \underset{\varepsilon \in I\!R}{argmax}\ \Delta i(b_\varepsilon^{X_m}, n_j). \tag{2.19}$$

Definition 2.2.6. *Let $\widehat{\mathbf{X}} \subseteq \mathbf{X}$ be contained in node n_j. Let $b_{\varepsilon_m^*}^{X_m}$ be optimal binary partitioning by X_m for $m \in \hat{r}$. Overall optimal partitioning b^* is found as*

$$b^* = \underset{m \in \{1,...,r\}}{argmax}\ \Delta i(b_{\varepsilon_m^*}^{X_m}, n_j). \tag{2.20}$$

When training the decision tree classifier, each node n_j is assigned a class based on the samples it contains. If the majority belongs to C_0 (C_1), node is assigned label C_0 (C_1) and every sample in the node is considered to be from C_0 (C_1) class. Let N_0 be number of samples from C_0 in the node n_j and let N_1 be number of samples from C_1 in the node n_j. Let $N_0 > N_1$. In addition to the assigned label, we consider that sample contained in the node n_j is from C_0 with $\dfrac{N_0}{N_0 + N_1}$ probability (similarly, we can reproduce the approach for scenario, when $N_1 > N_0$).

Building and randomizing the ensemble

When thinking about the best possible decision tree that can be trained, naturally, the idea of *fully developed tree* emerges. By fully developed tree we understand a tree with nodes containing only one sample. Hence, the class C_0 or C_1 is assigned with 100 % accuracy over the training set. However, such tree would be of dubious classification quality over data not included in the training set. This behaviour pushes to introduce stopping rules in order to control the tree growth, e.g. maximum tree depth, minimum samples in the leaves, etc. Inevitably, this fact limits the possible number of Ω space partitioning. To increase this number, further maximization of information extraction from the data avoiding over-training at the same time, different ideas of using multiple classifiers were proposed since the second half of 20^{th} century. E.g., boosted decision trees, ensemble linked using AdaBoost algorithm described in [1], are widely used in high energy physics community for data analysis today. In order to test these approaches, random forest, will be applied and tested over D^0 decay data. Random forest is an ensemble of trees grown over the training sets formed from original training set using bagging.

Definition 2.2.7. *Let $M \in \mathbb{N}$ and let L_{train} exists. Bagging is defined as producing $L_{train\ 1}, ..., L_{train\ M}$ sets, such that for $m \in \hat{M}$, $|L_{train\ m}| = n'$ for any $n' \in \mathbb{N}$. Elements of $L_{train\ m}$ are drawn from L_{train} uniformly randomly with replacement.*

When the ensemble is trained and fixed, each sample is classified separately by every tree in the forest. Finally, the sample is assigned to a class C_i, $i \in \{0, 1\}$ based on the majority of the votes of trees. Since every tree in the ensemble is grown over the very similar training set and optimal partitioning of the data contained in each node is deterministic, this would produce a forest of N similar trees. To avoid such situation and also to enable sub-optimal partitioning in the nodes to be used, randomness is introduced when looking for the optimal partitioning b^*. Optimization over $m \in \{1, ..., r\}$ is not allowed. Instead, random subset $M \subseteq \{1, ..., r\}$ is selected and optimization is performed for $m \in M$. Exploitation of sub-optimal partitioning and difference in trees architecture is ensured.

Hyper-parameters to be optimized

As mentioned in the previous two sections, when training tree or ensemble of trees, some hyper-parameters are to be selected. In order to perform optimal (or at least sub-optimal) selection automatically, k-fold cross-validation is used with specific metrics. Set of the parameters achieving the best average result over k-folds is selected as optimal. Below, the list of parameters optimized for D^0 classification problem is provided:

- Number of trees in the forest N
- Tree depth d
- Impurity measure (gini or entropy).

Optimization of selection of $M \subseteq \{1, ..., r\}$, where $|M|$ is either rounded value $\log_2 r$ or rounded value $\sqrt{r}$ and elements of M are drawn uniformly randomly from set $\{1, ..., r\}$ could not be performed, since for D^0 problem $\log_2 r \sim \sqrt{r}$.

2.3 Deep neural networks

Universal Approximation Theorem

Since 1989, various and improved versions of one of the most important theorems of machine learning have been stated. The key outcome of these theorems was that a function of certain attributes may be approximated as partial result (first layer output) of machine learning model known as neural network. Such version is known as universal approximation theorem for arbitrary width of the network. More general versions of the theorem for arbitrary depth of the network have been stated as well. But one thing is common for all the cases: Although the existence of such approximation is guaranteed, the theorem does not provide any instructions how network architecture should be built. In this section, theory introduced in [2] will be summarized to support following sections describing convolutional and residual neural networks.

Multilayer perceptron

Definition 2.3.1. *Let $x \in \mathbb{R}$. Linear activation function f is defined as*

$$f(x) = x. \tag{2.21}$$

Sigmoid activation function f is defined as

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.22}$$

Hyperbolic tangent activation function f is defined as

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.23}$$

Rectified Linear Unit (ReLU) activation function f is defined as

$$f(x) = \max\{x, 0\}. \tag{2.24}$$

Parametric Rectified Linear Unit (PReLU) activation function f is defined as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ ax, & \text{otherwise,} \end{cases} \tag{2.25}$$

where $a \in \mathbb{R}$ is learnable parameter.

Definition 2.3.2. *Let $\mathscr{C}$ be a binary classifier. Let $L = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_n}, y_n)\} = (\mathbf{X}, \mathbf{y})$ be a learning set. Binary cross-entropy loss function l_{BCE} is defined over L as*

$$l_{BCE} = -\frac{1}{n} \sum_{j=1}^{n} y_j \log(\mathscr{C}(\mathbf{x_j})) + (1 - y_j) \log(1 - \mathscr{C}(\mathbf{x_j})). \tag{2.26}$$

Note 2.3.1. *For distribution $p_j \in \{y_j, (1 - y_j)\}$ and distribution $q_j \in \{\mathscr{C}(\mathbf{x_j}), (1 - \mathscr{C}(\mathbf{x_j}))\}$ lets define cross-entropy $H(p_j, q_j)$. Binary cross-entropy loss function l_{BCE} can be then rewritten as*

$$l_{BCE} = \frac{1}{n} \sum_{j=1}^{n} H(p_j, q_j). \tag{2.27}$$

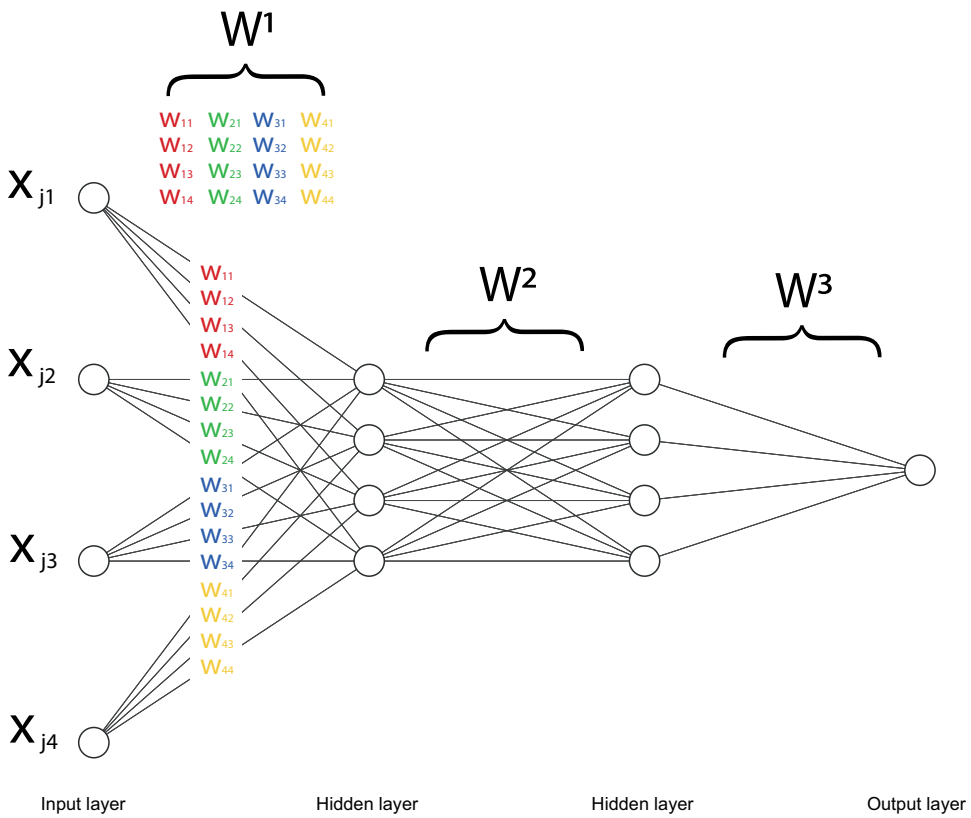


Figure 2.2: Multi-layer perceptron example.

Neural network model is a machine learning tool that can be represented as a set of elementary computational units (neurons) that may be connected. The connections enable the input information to be processed by one neuron and forwarded to the other. Each connection contains a weight, a real number, that scales the transferred information. Example of neural network model is displayed in Fig. 2.2.

Neurons are arranged in individual layers $\ell^1, ...\ell^k$. Layer ℓ^i contains N_i neurons. Layers are connected only with those layers and neurons situated directly in front of/behind them. The connection is mediated via real-valued matrix $\mathbf{W}^i$, which transforms the information contained in layer ℓ^i. Values contained in individual neurons may be also transformed by activation function f, e.g. as defined in Definition 2.3.1. In general, neurons may have different activation functions. In practice, the same activation function f_i is used for neurons in the layer ℓ^i. The introduced type of neural network is referred as multi-layer perceptron. Multi-layer perceptron may dispose of bias neurons, additional neurons containing trainable values that are added to the layer's output. However, such case can be represented by an input neuron containing value 1 for all the samples in the data set. Information inside the multi-layer perceptron travels as follows:

data contained in input layer:
$$\ell^1 = \mathbf{x} \tag{2.28}$$

transfer from i-th layer to $(i+1)$-th layer:
$$\ell^{i+1} = [f_{i+1}((\mathbf{W}^{i^T}\ell^i)_j)]_{j=1,...,N_{i+1}} \forall i \in \{1, ..., k-2\} \tag{2.29}$$

transfer from last inner layer to output layer:
$$\ell^k = [f_k((\mathbf{W}^{k-1^T}\ell^{k-1})_j]_{j=1,...,N_k}. \tag{2.30}$$

Prediction $\mathscr{C}(\mathbf{x})$ can be retrieved from ℓ^k by rounding the float contained in it for N_k=1. Otherwise it is found as $\underset{\{1,...,N_k\}}{argmax}\ \ell^k_j$.

Backpropagation

Although the previous section provided answer regarding the prediction system of the neural network, question regarding the network training emerged. It is clear, that the information transferred throughout the network is dependent on the set of weight matrices $\mathbf{W}^1, ..., \mathbf{W}^{k-1}$. When prediction of the set (or batch of the set) is provided, loss function, e.g. l_{BCE}, may be evaluated. To improve prediction ability, minimization of l_{BCE} value is necessary. Loss function l_{BCE} is composed function of elements of all matrices $\mathbf{W}^1, ..., \mathbf{W}^{k-1}$. Derivative with respect to all elements has to be provided to minimize the loss l_{BCE}. Therefore, solution decreasing computational difficulty is proposed: backpropagation algorithm. Let now w^i_{sj} be an element of $\mathbf{W}^i$, where $i \in \hat{k}$. Let also denote $\mathbf{W}^{iT} l^i$ as a^{i+1} (pre-activation value of the layer l^{i+1}). We will demonstrate finding partial derivative of l_{BCE} with respect to w^i_{sj}.

$$\frac{\partial l_{BCE}}{\partial w^i_{sj}} = \frac{\partial \ell_{BCE}}{\partial a^{i+1}_j} \frac{\partial a^{i+1}_j}{\partial w^i_{sj}}. \tag{2.31}$$

Let us denote

$$\delta^{i+1}_j \equiv \frac{\partial \ell_{BCE}}{\partial a^{i+1}_j}. \tag{2.32}$$

Also,

$$\frac{\partial a^{i+1}_j}{w^i_{sj}} = \ell^i_s, \tag{2.33}$$

therefore

$$\frac{\partial l_{BCE}}{\partial w^i_{sj}} = \delta^{i+1}_j \ell^i_s. \tag{2.34}$$

Finding δ^{i+1}_j is performed by starting from the last layer, for example, for $N_k = 1$, $j \in \{1\}$ and

$$\delta^k_1 = \frac{\partial l_{BCE}}{\partial a^k_1}. \tag{2.35}$$

For hidden layers we obtain

$$\delta^{i+1}_j = \frac{\partial l_{BCE}}{\partial a^{i+1}_j} = \sum_{r=1}^{N_{i+2}} \frac{\partial l_{BCE}}{\partial a^{i+2}_r} \frac{\partial a^{i+2}_r}{\partial a^{i+1}_j} = \sum_{r=1}^{N_{i+2}} \delta^{i+2}_r \frac{\partial a^{i+2}_r}{\partial a^{i+1}_j} = \sum_{r=1}^{N_{i+2}} \delta^{i+2}_r w^{i+2}_{jr} f'_{i+1}(a^{i+1}_j). \tag{2.36}$$

Therefore, δ^{i+1}_j is depedent on δ^{i+2}_j. Backpropagation algorithm provides us with a way how gradient of loss function may be estimated. Let $\mathbb{W}$ denote set of all elements of all matrices of weights in the network. Let l^j_{BCE} denote loss function evaluated for only one sample $\mathbf{x}_j$ from the learning set L, $j \in \hat{n}$. Let b denote indices of batch (learning set L subset for $b \subseteq 1, ..., n$). $\mathbb{W}$ may be optimized after each sample j that travels throughout the network as

$$\mathbb{W} \Longleftarrow \mathbb{W} - \nabla_{\mathbb{W}} l^j_{BCE}, \tag{2.37}$$

or after given batch travels through the network as

$$\mathbb{W} \Longleftarrow \mathbb{W} - \sum_{j \in b} \nabla_{\mathbb{W}} l^j_{BCE}. \tag{2.38}$$

When samples are drawn randomly from learning set L to be fed into the neural network, equation (2.37) is referred as stochastic gradient descent algorithm.

Optimizers

Stochastic gradient descent algorithm may be extended by learning rate γ which influences impact of computed gradient to the weight updating as

$$\mathbb{W} \Longleftarrow \mathbb{W} - \gamma \nabla_{\mathbb{W}} l^j_{BCE}, \tag{2.39}$$

or using batch as

$$\mathbb{W} \Longleftarrow \mathbb{W} - \gamma \sum_{j \in b} \nabla_{\mathbb{W}} l^j_{BCE}. \tag{2.40}$$

The value of learning rate is static throughout the whole training process. When learning rate is set too high, stochastic gradient descent may suffer from skipping the global minimum of the loss function. On the other hand, setting learning rate too low may lead to extremely slow algorithm convergence or not reaching the global minimum (or sufficient local minimum) at all since the optimizer would stay trapped inside insufficient local minimum. Therefore, it would be more convenient to use dynamic learning rate γ_t, that would reflect the loss function complexity during the process of weight updating. There exist two basic strategies for including the dynamic into learning rate

- exponential decay $\gamma_t = \gamma_0 \exp(-kt)$
- inverse decay $\gamma_t = \gamma_0 (1 + kt)^{-1}$,

where γ_0 is positive real number and fixed parameter set at the beginning. To avoid being trapped inside the local minimum of loss function, other strategies for $\mathbb{W}$ update were introduced:

- Optimization based on inertia parameter $\beta \in (0, 1)$ is defined by

$$\mathbb{W} \Longleftarrow \mathbb{W} - (\beta + 1)\gamma \nabla_{\mathbb{W}} l^j_{BCE}, \tag{2.41}$$

- AdaGrad optimization for initial value $A_i = 0$ is defined by

$$A_i \Longleftarrow A_i - \gamma \frac{\partial l_{BCE}}{\partial w}, \forall w \in \mathbb{W}, \tag{2.42}$$

  and then

$$w \Longleftarrow w - \frac{\gamma}{\sqrt{A_i + \epsilon}} \frac{\partial l_{BCE}}{\partial w}, \forall w \in \mathbb{W}, \tag{2.43}$$

  where ϵ is regularization parameter to avoid division by zero.

- RMSProp optimization for initial value $A_i = 0$ and decay parameter $\rho \in (0, 1)$ is defined by

$$A_i \Longleftarrow \rho A_i - (1 - \rho) \left(\frac{\partial l_{BCE}}{\partial w} \right)^2, \forall w \in \mathbb{W}, \tag{2.44}$$

  and

$$w \Longleftarrow w - \frac{\gamma}{\sqrt{A_i + \epsilon}} \frac{\partial l_{BCE}}{\partial w}, \forall w \in \mathbb{W}. \tag{2.45}$$

- Adam optimization for initial value $A_i = F_i = 0$ and decay parameters $\rho, \rho_f \in (0, 1)$ is defined by

$$A_i \Longleftarrow \rho A_i - (1 - \rho) \left(\frac{\partial l_{BCE}}{\partial w} \right)^2, \forall w \in \mathbb{W}, \tag{2.46}$$

  next

$$F_i \Longleftarrow \rho_f F_i - (1 - \rho_f) \frac{\partial l_{BCE}}{\partial w}, \forall w \in \mathbb{W}. \tag{2.47}$$

and

$$\gamma_t = \frac{\gamma_0 \left(\sqrt{1-\rho^t} \right)}{1-\rho_f^t}. \tag{2.48}$$

Finally,

$$w \Longleftarrow w - \frac{\gamma_t}{\sqrt{A_i + \epsilon}} F_i, \forall w \in \mathbb{W}. \tag{2.49}$$

Adam optimization is based on the L_2 norm. Variant of Adam with usage of infinity norm is referred to as Adamax.

Generalization techniques and hyper-parameter selection

For deep neural networks, the training may be influenced by the problems of vanishing/exploding gradient. For activation functions which absolute output values are between (0,1), multiplying their derivatives throughout many layers resolves in almost zero value due to number representation in the computer. Therefore, weights in the network are not updated accordingly. On the other hand, for activation functions with output grater than 1, multiplication of big numbers may resolved into NaN values in the algorithm. Therefore, wise selection of activation functions must be done. Training the neural network model is influenced by the initial weight values. Many approaches for smarter selection of appropriate weight distribution in the layers were introduced during the recent years. When implementing the neural network, each layer can be initiated differently.

Definition 2.3.3. *Let N_i be number of neurons in layer ℓ^i. Then, weights between layer ℓ^i and ℓ^{i+1} initiated by distribution*

$$U\left(-\sqrt{\frac{6}{N_i}}, \sqrt{\frac{6}{N_i}} \right) \tag{2.50}$$

are initiated with He uniform kernel.

Definition 2.3.4. *Let N_i, N_{i+1} be number of neurons in layer ℓ^i and ℓ^{i+1} respectively. Then, weights between layer ℓ^i and ℓ^{i+1} initiated by distribution*

$$U\left(-\sqrt{\frac{6}{N_i + N_{i+1}}}, \sqrt{\frac{6}{N_i + N_{i+1}}} \right) \tag{2.51}$$

are initiated with Glorot uniform kernel.

To increase generalization ability over the data, that model was not trained over, multiple generalization techniques may be introduced. For instance L_1/L_2 normalization penalized the loss function with L_1/L_2 norm of specified weights or outputs. This helps to suppress the influence of dominant connections of neurons formed over the training set.

For the better performance of the deep neural network model, standardization of the data set is recommended. There are multiple types of the standardization, that is usually fitted over training data and applied to whatever is fed to the network after the training.

Definition 2.3.5. *Let $\mathbf{x_i} = (x_{i1}, ..., x_{ir})$, where $x_{ij} \in \mathbb{R}$, $i \in \{1, ..., n\}$, $j \in \{1, ..., r\}$ and $r, n \in \mathbb{N}$ is observation of random variables $(X_1, ..., X_r)$. MinMax scaling is defined as*

$$x_{ij_{new}} = \frac{x_{ij} - m_j}{M_j - m_j}, \tag{2.52}$$

where $m_j = min\ \{x_{1j}, x_{2j}..., x_{nj}\}$, $M_j = max\ \{x_{1j}, x_{2j}..., x_{nj}\}$ for $\forall i \in \{1, ..., n\}$, $j \in \{1, ..., r\}$.

Definition 2.3.6. *Let* $\mathbf{x_i} = (x_{i1}, ..., x_{ir})$*, where* $x_{ij} \in \mathbb{R}$*,* $i \in \{1, ..., n\}$*,* $j \in \{1, ..., r\}$ *and* $r, n \in \mathbb{N}$ *is observation of random variables* $(X_1, ..., X_r)$*. Feature-wise standardization is defined as*

$$x_{ij_{new}} = \frac{x_{ij} - \bar{x}_j}{\sigma_j} \tag{2.53}$$

where $\bar{x}_j = \frac{\sum_{i=1}^{n} x_{ij}}{n}$ *a* $\sigma_j = \sqrt{\frac{\sum_{i=1}^{n} \left(x_{ij} - \bar{x}_j\right)^2}{n}}$ *pro* $\forall i \in \{1, ..., n\}, j \in \{1, ..., r\}$.

Another possibility, how the classifier's quality may be improved, is usage of batch normalization technique. This technique may be applied to the arbitrary neural network hidden layer after or before activation function is applied. Batch normalization performs feature-wise standardization of the batch of inputs it is fed. Therefore, the mean and variance of the batch coming from previous layer are learnable parameters of this transformation and influence form of backpropagation algorithm. The technique is used when coping with exploding/vanishing gradient problem. Other significant techniques for improving learning dynamics and generalization is dropout. During the training process, all available inputs of the training set are fed to the network multiple times repeatedly (epochs). When dropout of rate $d \in (0, 1)$ is introduced in between the layers, it forces the network to "forget" $100 \times d$ percent of the randomly selected connections (weights) of the layers it is placed between during one epoch. During following epoch, the "dead" connections are put back alive and $100 \times d$ percent of randomly selected connections are ignored again, until the learning is stopped by given stopping criterion.

2.4 Convolutional neural networks

Since convolutional neural network's theory was already described in [2], only brief summary will be provided in this section. For the situation, where the input type is matrix of tensor, taking into account the feature distribution in space is key driver for succesful classfication of such type of data. To put this idea into artificial neural network framework, discrete convolution plays a crucial role. Let F and G be real-valued matrices of arbitrary size. The product of their discrete convolution is defined by

$$(F \circledast G)_{k,l} = J_{k,l} = \sum_i \sum_j F_{i,j} \cdot G_{k-i,l-j}, \tag{2.54}$$

where the sum is performed using existing elements of F and G. Convolution result, J, is again a real valued matrix. Similarly, convolution for tensors may be defined. In this section, attributes of neural network type applicable to matrix/tensor inputs will be described.

Convolutional neural network architecture

Convolutional layer ℓ^i is characterized by set of filters (tensors or matrices) it contains. Elements of filters are learnable parameters during the training. When input is fed to the convolutional layer, it produces convolution of input with each filter in the layer. Such product of all convolutions is referred as feature map. The convolutional layer dimensionality is dependent on the number of filters it contains. The fact that convolution operation is translation invariant causes classificator invariance with respect to feature position within the input. Number of filters inside the convolutional layer determines the number of higher-level features extracted to the feature map. Feature map produced by previous convolutional layer may be input of further layer in order to extract even higher-level features. In convolutional neural network (CNN) architecture, usage of ReLU and PReLU activation functions is very common. The activation function may be applied to each element of given feature map individually. To reduce growing network dimensionality caused by usage of many filters in the process, introducing dimensionality-reduction techniques is beneficial. The most used are

- Strides: In general, filter with an input is moved during the convolution with step $s = 1$ and takes into account every element of input matrix/tensor. However, this step may be magnified and some elements of input matrix/tensor may be ommited.
- Pooling: Pooling layer P is special type of layer within the CNN architecture. Its dimensionality reflects the output dimensionality of previous layer. Let us denote output of previous layer $\ell^{\tilde{o}}$ as tensor $\tilde{o}$ with dimensionality $a \times b \times c$. Such tensor is an input of pooling layer P of dimensionality $p \times p \times c$ which may produce two types of outputs:
  - Max pooling: P is moving over input tensor with step size s and produces the output reduced to the dimensionality $(a-p)/s+1 \times (b-p)/s+1 \times c$, where each element is the maximum value of the input tensor area covered with P in a given step.
  - Average pooling: P is moving over input tensor with step size s and produces the output reduced to the dimensionality $(a-p)/s+1 \times (b-p)/s+1 \times c$, where each element is the average value of the input tensor area covered with P in a given step.

Padding

Moving filter of convolutional layer or padding layer P over input tensor is not well-defined instruction. To resolve the question, what input elements should be available for the operations in individual layer, padding is introduced. Let $p = d$ for pooling layer P or let filter within convolutional layer be squared with height d.

Following types of padding may be used:

- Half padding: Input is padded with $\frac{d-1}{2}$ zeros in its direct neighborhood.
- Valid padding: Input is not padded at all.
- Full padding: Input is padded with $d-1$ zeros in its direct neighborhood.

(a) Half padding

0	0	0	0	0	0
0	4	0	5	1	0
0	8	9	0	4	0
0	1	3	3	0	0
0	5	3	6	2	0
0	0	0	0	0	0

(b) Full padding

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	4	0	5	1	0	0
0	0	8	9	0	4	0	0
0	0	1	3	3	0	0	0
0	0	5	3	6	2	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 2.1: Padding for d=3.

When high-level features are extracted and contained as product of the final convolution, the tensor is flattened and used as an input of multi-layer perceptron which performs the classification based on high-level features. As for the DNN models, usage of generalization techniques, such as batch normalization or dropout inbetween the layers, is possible.

2.5 Residual neural networks

Networks of many layers may suffer from vanishing/exploding gradient problems. Until 2015, the problem have been solved with smarter application of weight (kernel) initialization of by batch normalization usage. However, in 2015, the concept of deep residual learning for image recognition was introduced. The idea behind residual learning lies in creating shortcuts in between the layers. For f_{i+1} being a ReLU function, a shortcut is defined as

$$\ell^{i+1} = f_{i+1}(\mathbf{W}^{i^T}\ell^i + \ell^{i-1}) \tag{2.55}$$

for matching dimensions of $\mathbf{W}^{i^T}\ell^i$ and ℓ^{i-1}. When the dimensions do not match, e.g. after the usage of strided convolutions, trainable mapping $\mathbf{W}^s$ (linear projection matrix) to the appropriate dimension for ℓ^{i-1} as $\mathbf{W}^s\ell^{i-1}$ is used. When the dimensions are matching and $\mathbf{W}^s$ is identity matrix, the shortcut has fixed weights that are not changed during the iterations of optimization. Otherwise, standard backpropagation algorithm is used. Adding a residual blocks to the deep learning model architecture provides easier-to-optimize model of low complexity and solves the problem with gradient degradation. Improved gradient flow enables model architect to add more layers and improve model performance quality at the same time. The skip connections proposed in [5] connect layers ℓ^i up to ℓ^{i+a}, where $a > 1$ and the architecture composed of many similar residual modules and only additive connections without trainable mapping $\mathbf{W}^s$ were present. In this thesis, model from [5] is not applied, however, feed forward convolutional network with additive residual module for $a = 1$ is built.

Chapter 3

Unsupervised machine learning

Models of binary classification described in the previous sections were trained over labeled learning set L. Very often, labels $\mathbf{y}$ for the learning samples are not available. Therefore, different approach than minimizing loss function or impurity measures must be introduced. Usage of spatial attributes of data $\mathbf{X}$ is common way how to perform classification into K classes represented by K clusters in r-dimensional Euclidean space, where $K, r \in \mathbb{N}$. Finding similarities is not only important for classification problems. Clustering data of $\mathbf{X}$ helps to join similar objects together, project them for better visualization of high dimensional input data or better understanding the data set. Such approach may be applicable for instance when finding similarities of events based on histograms of observed variables of the produced particles.

3.1 K-means

Let $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$, where $\mathbf{x_j} = \{x_{j1}, ..., x_{jr}\}$ and $x_{jr} \in \mathbb{R}$, be a set of vectors in r-dimensional Euclidean space. Assigning each vector from $\mathbf{X}$ to the one of the k classes (clusters) for $K \in \mathbb{N}$ is performed by finding set of vectors (centroids) $\{\mathbf{c_1}, ..., \mathbf{c_K}\}$ that would represent center of given class, and $\mathbf{x_j}$ is assigned to the class k^* if

$$k^* = \underset{k \in \{1,...,K\}}{argmin} \, ||\mathbf{x_j} - \mathbf{c_k}||^2. \tag{3.1}$$

K - means algorithm provides a way, how $\{\mathbf{c_1}, ..., \mathbf{c_K}\}$ may be found.

Definition 3.1.1. *Let $n, K \in \mathbb{N}$, $j \in \hat{n}$ and $k, k^* \in \hat{K}$. Let*

$$k^* = \underset{k \in \{1,...,K\}}{argmin} \, ||\mathbf{x_j} - \mathbf{c_k}||^2. \tag{3.2}$$

Let $\gamma_{jk} = 0$ for $k \neq k^*$ and $\gamma_{jk} = 1$ for $k = k^*$. Distortion measure J is defined as

$$J = \sum_{j=1}^{n} \sum_{k=1}^{K} \gamma_{jk} ||\mathbf{x_j} - \mathbf{c_k}||^2. \tag{3.3}$$

Values γ_{jk} and $\{\mathbf{c_1}, ..., \mathbf{c_K}\}$ are found by minimizing distortion measure J. In extreme case $K = n$ and $\mathbf{x_j} = \mathbf{c_j}$ for all $j \in \hat{n}$. However, such clustering would not bring any new insight to the problem. Therefore, K must be chosen based on prior (expert) information about the data. Distortion measure J is minimized by EM algorithm in two steps, E (expectation) and M (maximization). First k^* is computed. This determines the values of γ_{jk}.

Next, for γ_{jk} fixed, $\mathbf{c_k}$ is estimated by

$$\mathbf{c_k} = \frac{\sum_{j=1}^{n} \gamma_{jk} \mathbf{x_j}}{\gamma_{jk}}. \tag{3.4}$$

Convergence of K-means is guaranteed by convergence of EM algorithm as previously proved in [1]. Initialization of $\{\mathbf{c_1}, ..., \mathbf{c_K}\}$ for the first iteration may be random (e.g. K samples from $\mathbf{X}$ are selected). Although K-means is able to cluster data into subsets based on their spatial similarity, it does not provide direct answer if data from cluster of $\mathbf{c_a}$ are more similar to data from cluster of $\mathbf{c_b}$ or data from cluster of $\mathbf{c_d}$, especially in case for $\|\mathbf{c_a} - \mathbf{c_b}\|^2 \sim \|\mathbf{c_a} - \mathbf{c_d}\|^2$, for $a, b, d \in \hat{K}$. For applications, where such relationships need to be estimated, self organizing maps may provide answer to this question.

3.2 Self organizing maps

Let $X_1, X_2, ..., X_n$ be random variables of continuous probability distributions $f_1, f_2, ..., f_n$. Let $h_1, h_2, ..., h_n$ be a normalized discrete representations (histograms) of $f_1, f_2, ..., f_n$ from the experimental measurement, where $X_1, X_2, ..., X_n$ are observed. Let each histogram be of r bins. For histogram h_i, where $i \in \hat{n}$, value (count) in bin $j \in \hat{r}$ will be denoted as p_{ij}. Let $\mathbf{p}_i = (p_{i1}, ..., p_{ir})$. We aim to find an ordering function $\phi : \mathbf{p}_i \longrightarrow \mathbb{R}$, which provides us with ordering of $\phi(\mathbf{p}_1)$,...,$\phi(\mathbf{p}_n)$ in terms of $<, >, =$. By transforming $h_1, h_2, ..., h_n$ to the $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n$ we are able to perform ordering of histograms themselves. The ordering will be found using special type of artificial neural network: self organizing maps. Let $N \gg r$ for $N \in \mathbb{N}$. Let $s_1, ..s_N$ be representation of N neurons within a layer S_N. Let every neuron from $s_1, ..s_N$ be connected with r-dimensional input layer I_r. At the same time, let all neurons from $s_1, ..s_N$ be connected via a grid. Example of such network/map for $r = 7$, where $\mathbf{p}_i$ are fed to the input layer $I_{r=7}$, is visualized in Fig. 3.1.

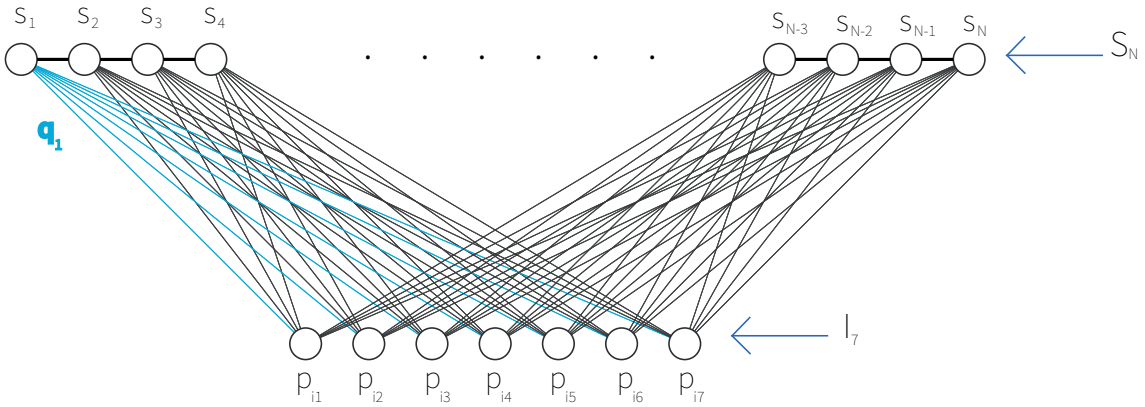


Figure 3.1: Self organizing map for $r = 7$

The connection between j-th input neuron, where value p_{ij} is placed, and neuron in layer s_k may be represented as a real number q_{kj}. Let $\mathbf{q}_k = (q_{k1}, ..., q_{kr})$ be real-valued r-dimensional vector. When $\mathbf{p}_i$ is fed to the map, and $\mathbf{q}_1^0, ..\mathbf{q}_N^0$ are initialized randomly for the first iteration $t = 0$, value of neuron s_k^0 is given as a dot product

$$s_k^0 = \mathbf{p}_i \cdot \mathbf{q}_k^0 \tag{3.5}$$

Next, $\phi(\mathbf{p}_i)^0$ is calculated as

$$\phi(\mathbf{p}_i)^0 = max\,\{s_1^0, ..., s_N^0\} = s_{k^*}^0. \tag{3.6}$$

Neuron $s_{k^*}^0$ is recognized as best matching at $t = 0$, since the similarity between $\mathbf{p}_i$ and $\mathbf{q}_k^0$ was maximized. However, such ordering would be purely dependent on random initialization of $\mathbf{q}_1^0, ..\mathbf{q}_N^0$ at $t = 0$.

Therefore, system for updating of $\mathbf{q}_1^t, ..\mathbf{q}_N^t$ to $\mathbf{q}_1^{t+1}, ..\mathbf{q}_N^{t+1}$ must be introduced. Updating weights is also conditioned by the requirement, that in case when $\phi(\mathbf{p}_i)^t = s_{k^*}^t$, $\phi(\mathbf{p}_m)^t = s_{k^*+1}^t$ and $\phi(\mathbf{p}_l)^t = s_{k^*-1}^t$, histograms h_m, h_l are the most similar to h_i. However h_m is more similar to the h_i than to the h_l. In the same manner, h_l is more similar to the h_i than to the h_m. This attribute is guaranteed by the concept of competitive learning. When $\mathbf{q}_{k^*}^t$ is updated, weight vectors corresponding to the rest of the neurons are updated as well based on the neurons' closeness to $s_{k^*}^t$. The closer to the $s_{k^*}^t$ the bigger the update impact. The closeness is mediated using the grid between neurons (Euclidean distance of the value carried by the units). The iteration algorithm for weight update is given by the equation 3.7

$$\mathbf{q}_k^{t+1} = \mathbf{q}_k^t + \gamma^t[\mathbf{p}_t - \mathbf{q}_k^t]K^t(k^*, k), \tag{3.7}$$

where $K^t(k^*, k)$ is a smoothing kernel reflecting distance of $s_{k^*}^t$ and s_k^t at iteration/time t as

$$K^t(k^*, k) \qquad \begin{cases} = 1, & \text{if } k^* = k, \\ < 1, & \text{otherwise.} \end{cases} \tag{3.8}$$

At time t, sample $\mathbf{p}_i$ is drawn randomly from the set $\{\mathbf{p}_1, ..., \mathbf{p}_n\}$ and values of $\mathbf{q}_1^t, ..\mathbf{q}_N^t$ are updated. The algorithm performs given amount of steps $t \in \{1, ..., T\}$, where $T \in \mathbb{N}$. Since self organizing maps allow forming the projection of r-dimensional input space to the $\mathbb{R}$, clustering over $\mathbb{R}$ is rather trivial. Moreover, samples projected to such clusters may be still compared (ordered) using ordering function ϕ.

Chapter 4

Experimental results

In this chapter, experimental results for D^0 decay and QCD transitions studies are presented. In both cases, experiments were implemented using Python 3 programming language with the following libraries: Keras with TensorFlow backend, matplotlib, numpy, pandas, scikit-learn, seaborn, statsmodels and uproot with required dependencies. To obtain results in reasonable time, HELIOS cluster at the Department of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague was used for computation tasks. Data for D^0 decay experiment were obtained from the STAR experiment and physics-wise pre-processing was performed by Ing. L. Kramárik from the Department of Physics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague. Data for QCD transitions experiment were simulated by Ing. J. Cimerman from the Department of Physics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague.

4.1 D^0 decay data

In this section, the problem of identifying D^0 decay products as introduced in section 1.1 is presented. The classification was performed based on the subset of features available in Table 1.1. Before starting the training of optimized classifiers, question about suitable background class representation as proposed in 1.1 needed to be resolved. Since the background could have been represented by non-signal unlike-sign pairs or like-sign pairs, a test of distribution homogeneity of such populations was performed. The test was performed, after pre-cuts from Table 1.1 were applied to the data. Since the training of classifiers described in this chapter will be performed over the samples from narrow p_T value range, homogeneity was tested within the same p_T intervals. As described in Fig. 4.1, 4.2 and 4.3, homogeneity for the features was accepted on significance level 0.01 when Kolmogorov-Smirnov test was applied to the population of 2000 samples randomly selected from each type of background representation. Furthermore, during the initial phase of analysis, it was observed that representing the background as non-signal unlike-sign pairs resulted in almost the same classification quality as using like-sign pairs as background representation. Therefore, based on this empirical experience, only unlike-sign background representation was eventually used for the experiments described in the following sections. However, on the same significance level, the homogeneity was not always accepted (Fig. 4.4, 4.5, and 4.6) when comparing HIJING background representation and background representation from [2] (data). Specifically, the homogeneity was rejected for DCA_K distribution in $1<p_T<2$ GeV/c, for DCA_π, DCA_K, l_{decay}, $DCA_{\pi,K}$, DCA_{D^0} and $\cos\theta^*$ distributions in $2<p_T<3$ GeV/c and for DCA_π, DCA_K, l_{decay}, $DCA_{\pi,K}$, DCA_{D^0}, $\cos\theta$, $\cos\theta^*$ and p_T distributions in $3<p_T<5$ GeV/c. Still, the similarity was sufficient for satisfying application of the existing classifier trained in [2] to HIJING data transformed with feature-wise standardization fitted over training set in [2]. Details of the application are listed in subsection 4.1.4.

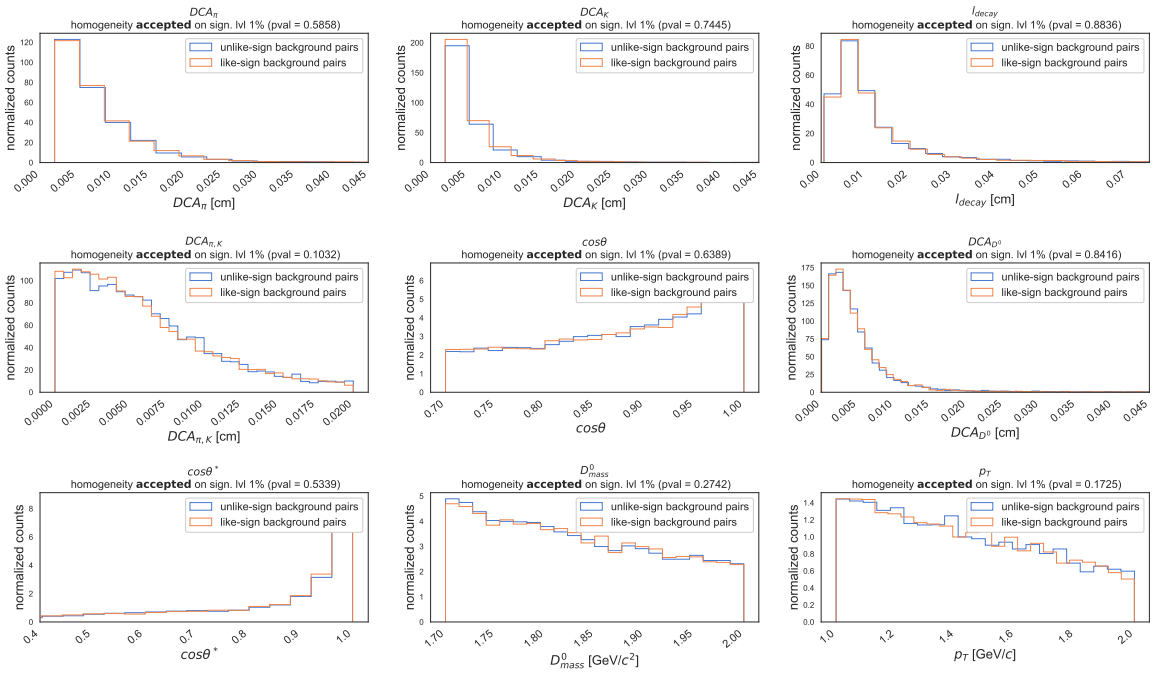


Figure 4.1: Homogeneity test for possible background class representation in $1 < p_T < 2$ GeV/c.

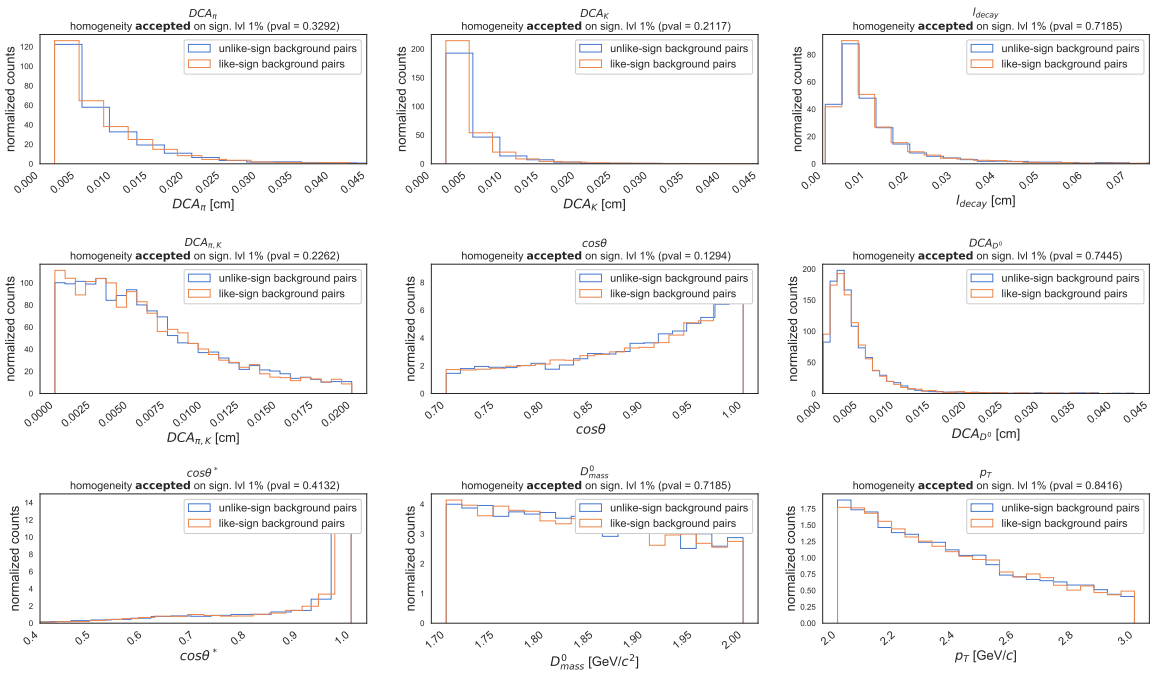


Figure 4.2: Homogeneity test for possible background class representation in $2 < p_T < 3$ GeV/c.

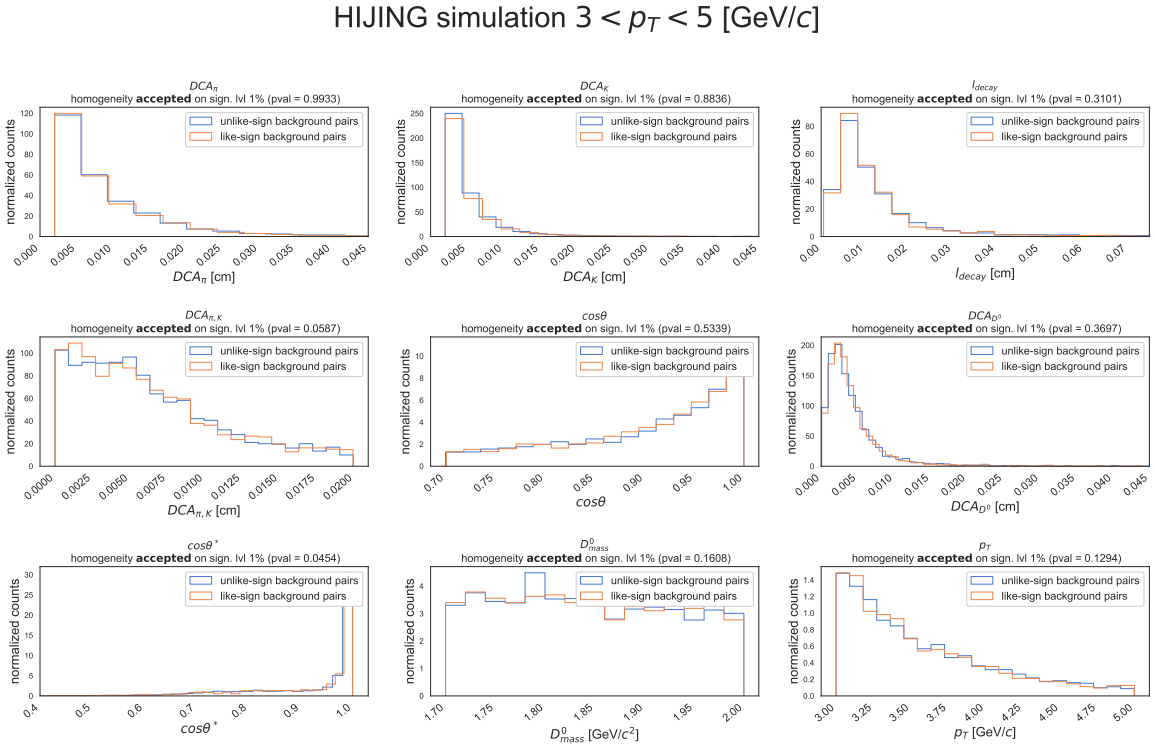


Figure 4.3: Homogeneity test for possible background class representation in $3 < p_T < 5$ GeV/c.

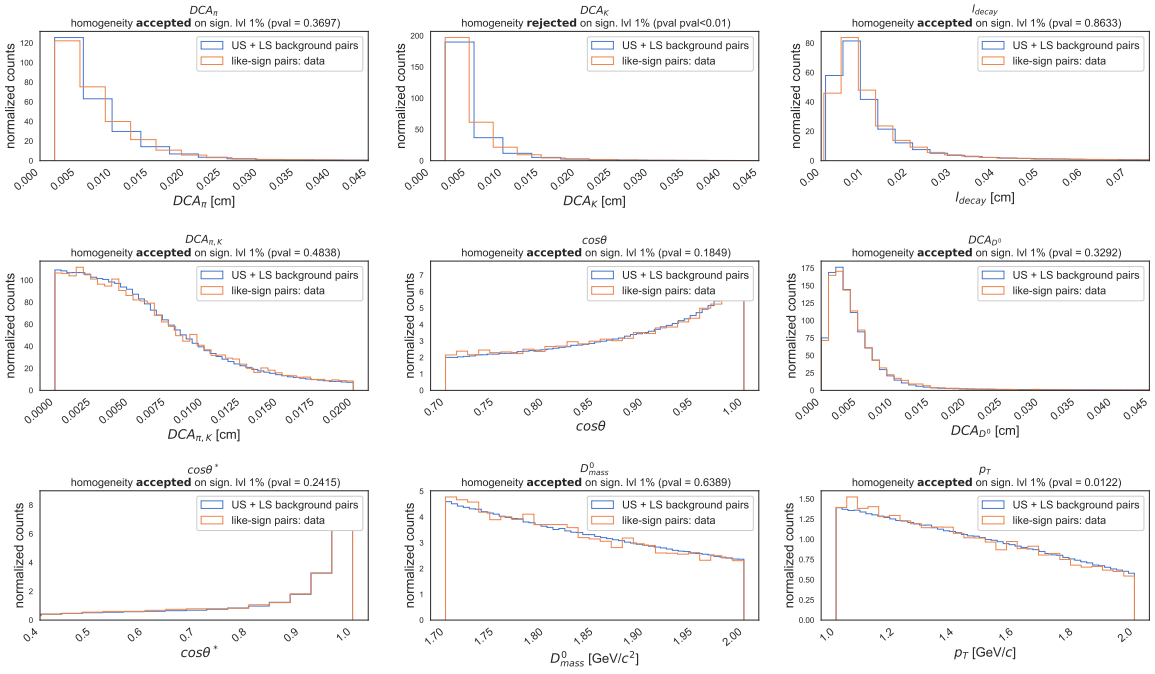


Figure 4.4: Homogeneity test for HIJING background class representation and measured data represented background in $1 < p_T < 2$ GeV/c.

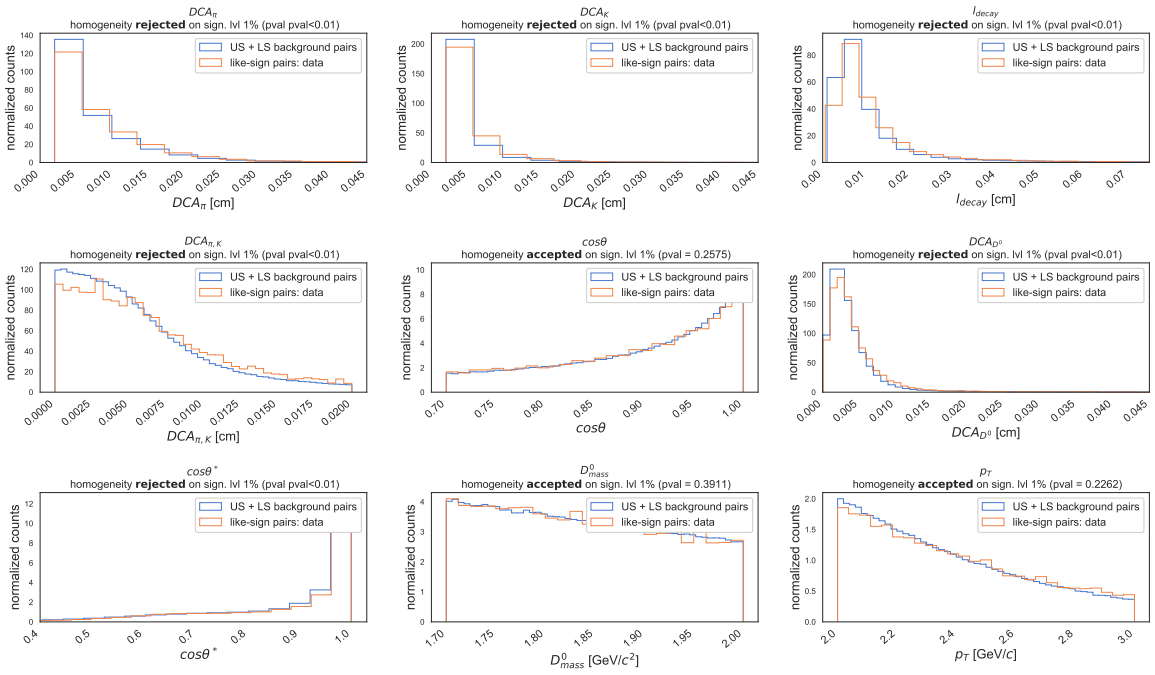


Figure 4.5: Homogeneity test for HIJING background class representation and measured data represented background in $2 < p_{\mathrm{T}} < 3$ GeV/c.

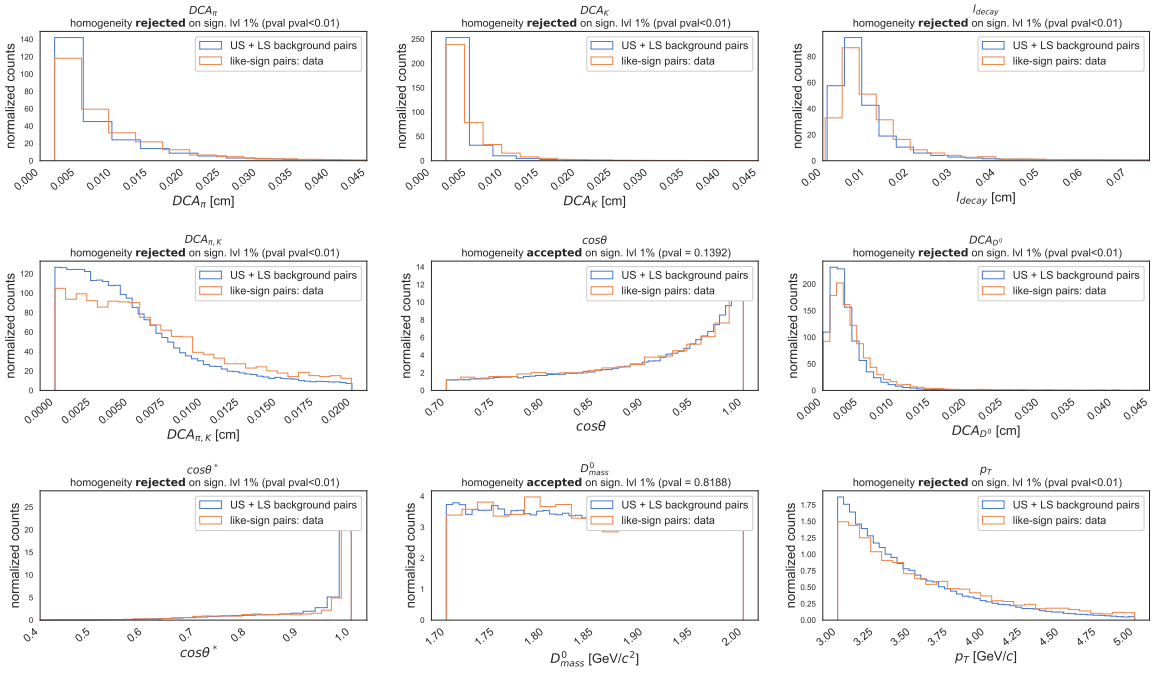


Figure 4.6: Homogeneity test for HIJING background class representation and measured data represented background in $3 < p_{\mathrm{T}} < 5$ GeV/c.

4.1.1 Random forests (RF)

Random forest classifier was applied to HIJING simulation in two different setups. First setup starts with dividing the data into training and test set (6:4 ratio) after application of pre-cuts described in Table 1.1. Next, each of these two parts was subdivided by the value of p_T into five disjoint p_T bins: 0–1 GeV/c, 1–2 GeV/c, 2–3 GeV/c, 3–5 GeV/c and 5–8 GeV/c. For each p_T bin, optimized random forest was trained. But before that, *balancing* of the data set was performed. Number of samples of each class is estimated in the training set. Size of the class with the lower number of samples is used as maximum allowed size of the class in the training set of given p_T bin. Appropriate number of samples from the second class is then removed using random selection, until both of the classes are of the same size. The same procedure was performed for the test set. The hyper-parameter space for optimization of the classifier is given by Table 4.5 and is the same for all p_T bins. During the training phase, training set of given p_T bin is used and the 3-fold cross-validation is applied to train random forest for each combination of hyper-parameters. Classifier with the highest AUC value is selected as the optimized one and tested over test part of matching p_T bin value data. AUC and other binary classification metrics are monitored as well (P, R, NPV, TNR, ACC and AUC) and their evaluation over test set is described by Table 4.1. The parameters of classifier with the highest AUC are for each p_T bin described in Table 4.3. For the second experiment setup, the whole process described above is applied in the same manner, but pre-cuts from Table 1.1 are not applied this time. In [1], we have already confirmed, that application of pre-cuts has positive effect on the classifier quality. However, pre-cuts are chosen empirically and may differ for different applications in physics. Therefore, comparison of classifiers (not only hyper-parameters) for non-pre-cut data is required as well. For the second setup, optimal parameters are presented in Table 4.4 and performance of optimized classifiers in Table 4.2. We explain the decrease of P and TNR in Table 4.1 for p_T bin 5–8 GeV/c by smaller amount of data in test set. Impurity measure selected as optimal for each scenario was entropy, however there seems to be no trend with respect to increasing p_T value in selection of number of tree and maximum depth of the ensemble. However, in the scenario, where pre-cuts are not applied, optimization selected deeper trees within the ensemble, since more complex input space partition was needed.

p_T bin id	p_T [GeV/c]	P	R	NPV	TNR	ACC	AUC
bin0	0–1	0.74	0.68	0.70	0.76	0.72	0.81
bin1	1–2	0.81	0.73	0.75	0.83	0.78	0.87
bin2	2–3	0.89	0.82	0.83	0.89	0.86	0.94
bin3	3–5	0.91	0.85	0.86	0.92	0.89	0.95
bin4	5–8	0.89	0.92	0.91	0.89	0.90	0.96

Table 4.1: Evaluation of optimized random forest classifiers over test set, pre-cuts applied.

p_T bin id	p_T [GeV/c]	P	R	NPV	TNR	ACC	AUC
bin0	0–1	0.68	0.60	0.64	0.72	0.66	0.73
bin1	1–2	0.72	0.68	0.70	0.74	0.71	0.79
bin2	2–3	0.73	0.76	0.75	0.72	0.74	0.82
bin3	3–5	0.84	0.79	0.80	0.85	0.82	0.90
bin4	5–8	0.89	0.82	0.84	0.90	0.86	0.94

Table 4.2: Evaluation of optimized random forest classifiers over test set, pre-cuts not applied.

In accordance with key outcomes of [1] and [2], the overall classification quality decreases when pre-cuts are not applied and increases with p_T value as presented in Fig. 4.7 where models achieved higher

AUC when p_T value was also higher. As visualized in Fig. 4.8, relatively higher rates of FP and FP are present for lower p_T bins with respect to test set size.

p_T bin id	p_T [GeV/c]	number of trees	maximum depth	impurity measure
bin0	0–1	1200	10	entropy
bin1	1–2	600	13	entropy
bin2	2–3	1000	12	entropy
bin3	3–5	200	10	entropy
bin4	5–8	200	7	entropy

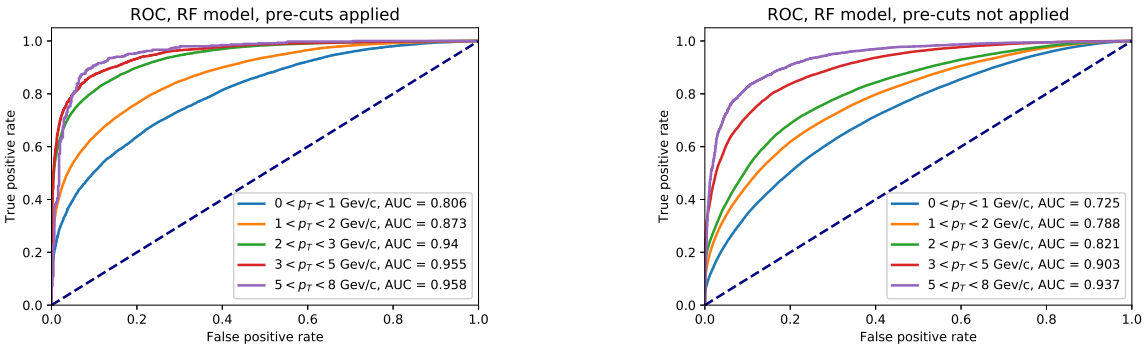
Table 4.3: Parameters of optimized random forest classifiers, pre-cuts applied.

p_T bin id	p_T [GeV/c]	number of trees	maximum depth	impurity measure
bin0	0–1	1200	15	entropy
bin1	1–2	600	15	entropy
bin2	2–3	1000	12	entropy
bin3	3–5	1200	12	entropy
bin4	5–8	200	7	entropy

Table 4.4: Parameters of optimized random forest classifiers, pre-cuts not applied.

parameter type	optimization range
number of trees	200, 400, 600, 800, 1000, 1200
maximum depth	5, 7, 10, 12, 15, 17, 20
impurity measure	gini, entropy

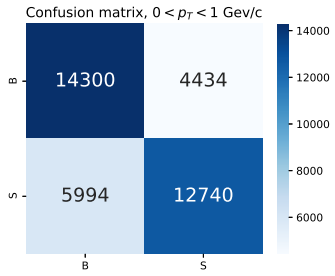
Table 4.5: Hyper-parameter space definition for random forest optimization.



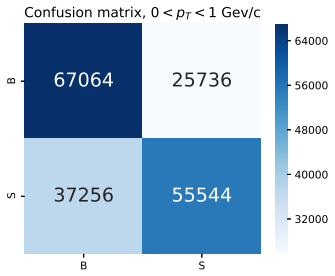
(a) ROC, pre-cuts applied.

(b) ROC, pre-cuts not applied.

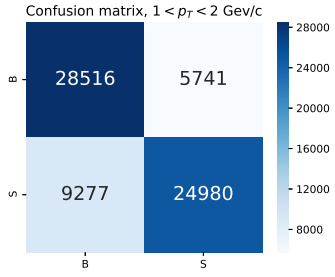
Figure 4.7: RF model, comparison of ROC and AUC for scenario with and without cut application, evaluated over test set.



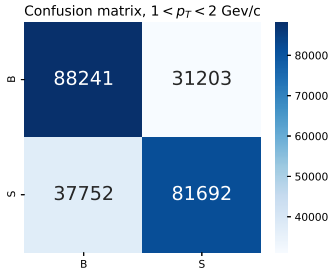
(a) p_{T} bin 0–1 GeV/c, pre-cuts applied.



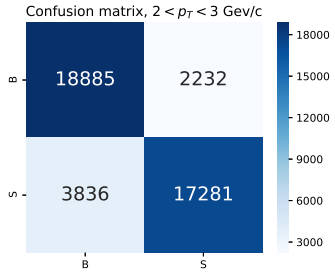
(b) p_{T} bin 0–1 GeV/c, pre-cuts not applied.



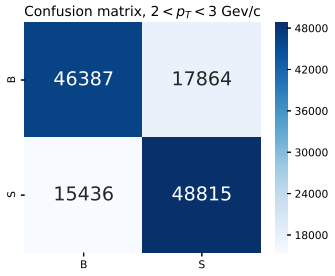
(c) p_{T} bin 1–2 GeV/c, pre-cuts applied.



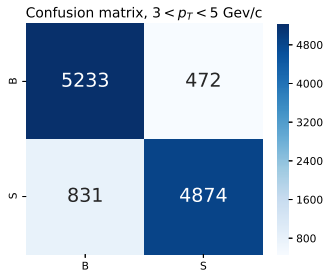
(d) p_{T} bin 1–2 GeV/c, pre-cuts not applied.



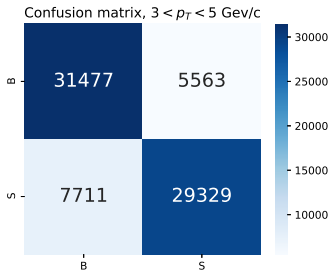
(e) p_{T} bin 2–3 GeV/c, pre-cuts applied.



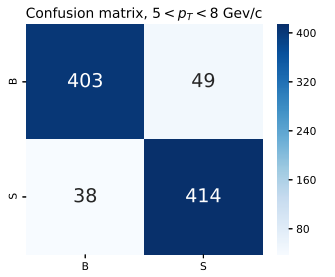
(f) p_{T} bin 2–3 GeV/c, pre-cuts not applied.



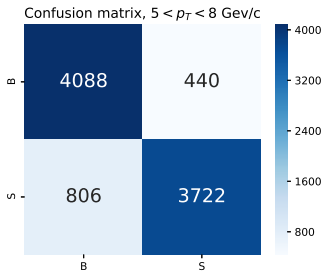
(g) p_{T} bin 3–5 GeV/c, pre-cuts applied.



(h) p_{T} bin 3–5 GeV/c, pre-cuts not applied.



(i) p_{T} bin 5–8 GeV/c, pre-cuts applied.

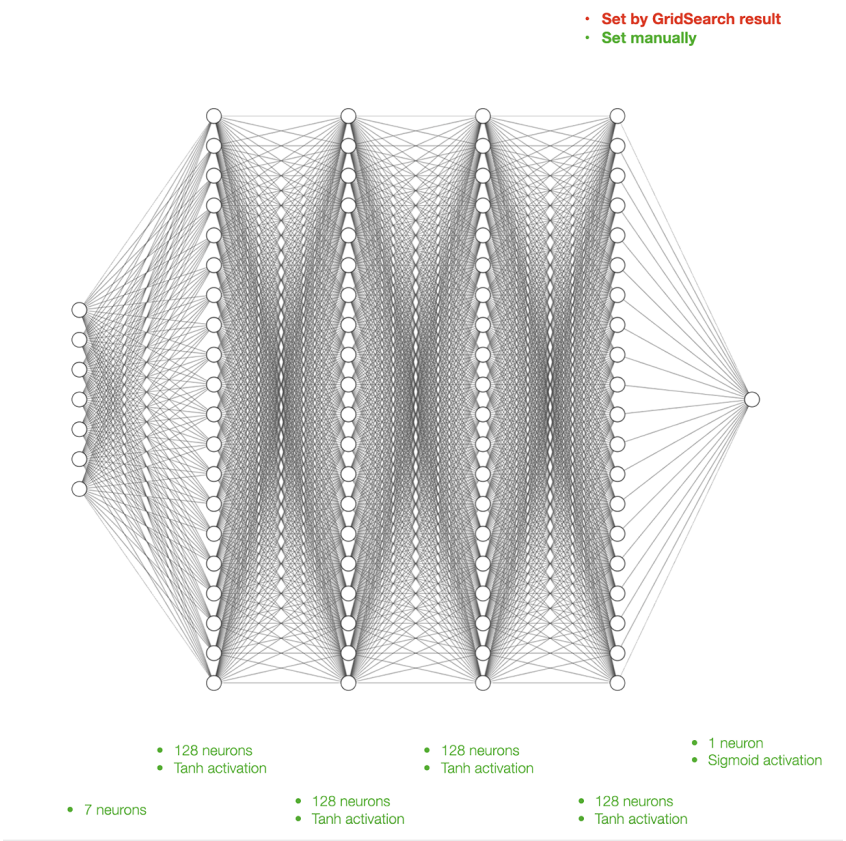


(j) p_{T} bin 5–8 GeV/c, pre-cuts not applied.

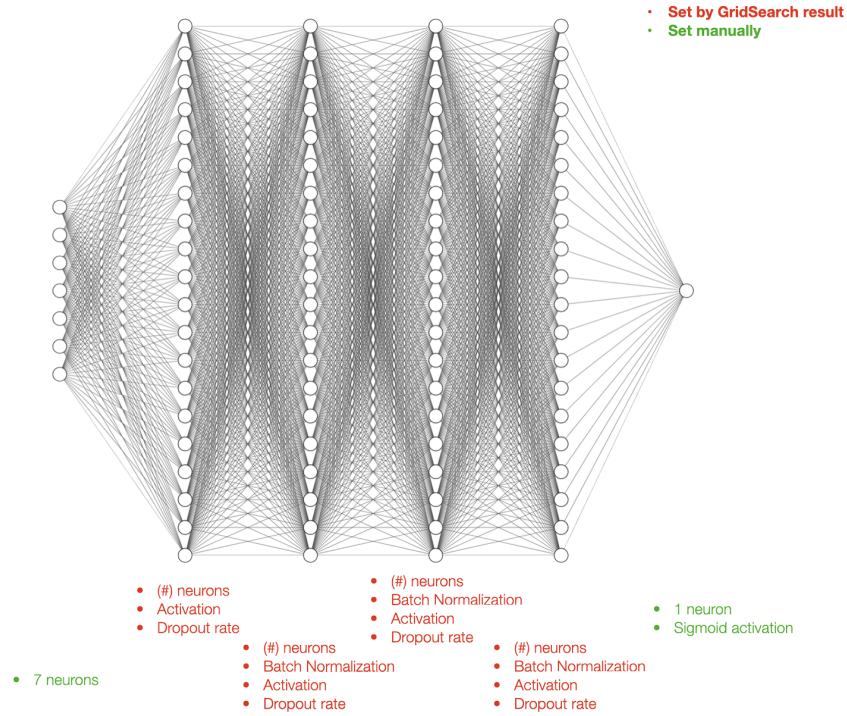
Figure 4.8: RF model, comparison of confusion matrices for scenario with and without cut application, evaluated over test set.

4.1.2 Deep neural network (DNN)

In the similar manner as random forests, deep neural networks are optimized over train set and evaluated over the test set. However, slightly different modifications are performed inside the pipeline. As for the random forests case, two setups exist for deep neural models as well: first for the data set under pre-cuts from Table 1.1, second for no such pre-processing. Unlike for random forests, data are divided into three subsets: training, validation and test in 6:2:2 ratio. Each of these three parts was subdivided into 5 p_{T} bins (same as in the random forest approach). For each p_{T} bin, *balancing* of the training, validation and test set was performed since neural networks would not be able to converge over the extremely unbalanced training set. In [2], the architecture and the hyper-parameters were not optimized but selected empirically. In the architecture we are introducing for the purposes of this thesis, high rate of regularization techniques is applied (drop out rate, batch normalization), due to increased similarity of the inputs available for training (simulation). In Fig. 4.9, fixed and trainable parameters of the deep neural network model are described and model from [2] is compared to the current architecture. The architecture was derived from the architecture used in [2]. Before training or applying the classifiers, feature-wise standardization was fitted over training set of given p_{T} bin and applied to the corresponding validation and test set. The hyper-parameter space for optimization of the classifier is given by Table 4.6 and again it is the same for all p_{T} bins. Hyper-parameter ranges were selected empirically. The training phase over given p_{T} bin was performed in two steps. During the first step, for each combination of hyper-parameters, deep neural network model is trained for 70 epochs. During the training, values of training and validation ACC is monitored. Loss function l_{BCE} evaluated over training and validation set is monitored as well. The parameters of the model with the highest AUC are then used to perform the second step, where the training of the model starts all over again with parameters values fixed. The training is now performed for 500 epochs and is stopped only in case, that validation ACC was not improved during the last 60 epochs (early stopping rule). Since the validation set was already used for hyper-parameter selection and influences the time classifier is trained for, binary classification metrics (AUC, P, R, NPV, TNR, ACC and AUC) are evaluated over the test set only. Values of optimal hyper-parameters of given p_{T} bin are described in Table 4.7 (pre-cuts applied) and 4.8 (pre-cuts not applied). Classifiers performance is described in Table 4.9 (pre-cuts applied) and 4.10 (pre-cuts not applied). Additionally, in Fig. 4.10 and 4.11, the second step of optimized classifier's training is visualized by the evolution of the accuracy and the loss function value (binary crossentropy l_{BCE}.) When comparing speed of loss minimization, there was no major diference for over the pre-cut data and non pre-cut data. However, training the model over data where pre-cuts were not applied took more epochs. The slowest loss function minimization was to be seen in the last p_{T} bin. We assume that this effect is caused by smaller training set size. Application of pre-cuts resulted in classifiers' performance improvement (the same as for random forests) and as expected, for higher p_{T} values classifiers achieved better quality as well. As experienced with random forests over data under pre-cuts, some binary classification metrics decreased for p_{T} bin 5–8 GeV/c (TNR, AUC). The same as in previous case we explain such behavior as statistical fluctuation caused by unsufficient number of samples in test set for such high p_{T} value. Unlike optimization result for random forest, optimized parameters for all p_{T} bins for both scenarios of pre-cut usage are rather similar as presented in Table 4.7 and 4.8.



(a) DNN model proposed in [2].



(b) DNN model proposed in this thesis.

Figure 4.9: Comparison of DNN models proposed in [2] and in this thesis

parameter type	optimization range
activation function	ReLU, tanh
learning rate	0.000001,0.000005, 0.00001
dropout	0.2, 0.3, 0.4
kernel	lecun uniform, he normal, he uniform
hidden layer width	64, 128, 256
optimizer	adam, SGD, adamax
batch	64, 128

Table 4.6: Hyper-parameter space definition for deep neural network optimization.

p_T bin id	p_T [GeV/c]	activation	learn. rate	dropout	kernel	optimizer	batch	hid. layer width
bin0	0–1	ReLU	1E-05	0.2	adam	he normal	64	256
bin1	1–2	ReLU	1E-05	0.2	adam	he uniform	64	256
bin2	2–3	ReLU	1E-05	0.2	adam	lecun uniform	64	256
bin3	3–5	ReLU	1E-05	0.2	adam	lecun uniform	64	256
bin4	5–8	ReLU	1E-05	0.2	adam	lecun uniform	64	256

Table 4.7: Parameters of optimized deep neural network classifiers, pre-cuts applied.

p_T bin id	p_T [GeV/c]	activation	learn. rate	dropout	kernel	optimizer	batch	hid. layer width
bin0	0–1	ReLU	1E-05	0.2	adam	he normal	64	256
bin1	1–2	ReLU	1E-05	0.2	adam	lecun uniform	64	256
bin2	2–3	ReLU	1E-05	0.2	adam	lecun uniform	64	256
bin3	3–5	ReLU	1E-05	0.2	adam	lecun uniform	64	256
bin4	5–8	tanh	1E-05	0.2	adam	lecun uniform	64	256

Table 4.8: Parameters of optimized deep neural network classifiers, pre-cuts not applied.

p_T bin id	p_T [GeV/c]	P	R	NPV	TNR	ACC	AUC
bin0	0–1	0.73	0.68	0.70	0.75	0.72	0.79
bin1	1–2	0.77	0.72	0.74	0.79	0.75	0.83
bin2	2–3	0.83	0.80	0.81	0.84	0.82	0.90
bin3	3–5	0.85	0.87	0.87	0.85	0.86	0.93
bin4	5–8	0.82	0.91	0.90	0.80	0.86	0.91

Table 4.9: Evaluation of optimized deep neural network classifiers over test set, pre-cuts applied.

p_T bin id	p_T [GeV/c]	P	R	NPV	TNR	ACC	AUC
bin0	0–1	0.67	0.58	0.63	0.71	0.65	0.70
bin1	1–2	0.73	0.63	0.68	0.76	0.70	0.76
bin2	2–3	0.77	0.73	0.74	0.78	0.75	0.83
bin3	3–5	0.82	0.80	0.80	0.82	0.81	0.89
bin4	5–8	0.83	0.88	0.87	0.82	0.85	0.92

Table 4.10: Evaluation of optimized deep neural network classifiers over test set, pre-cuts not applied.

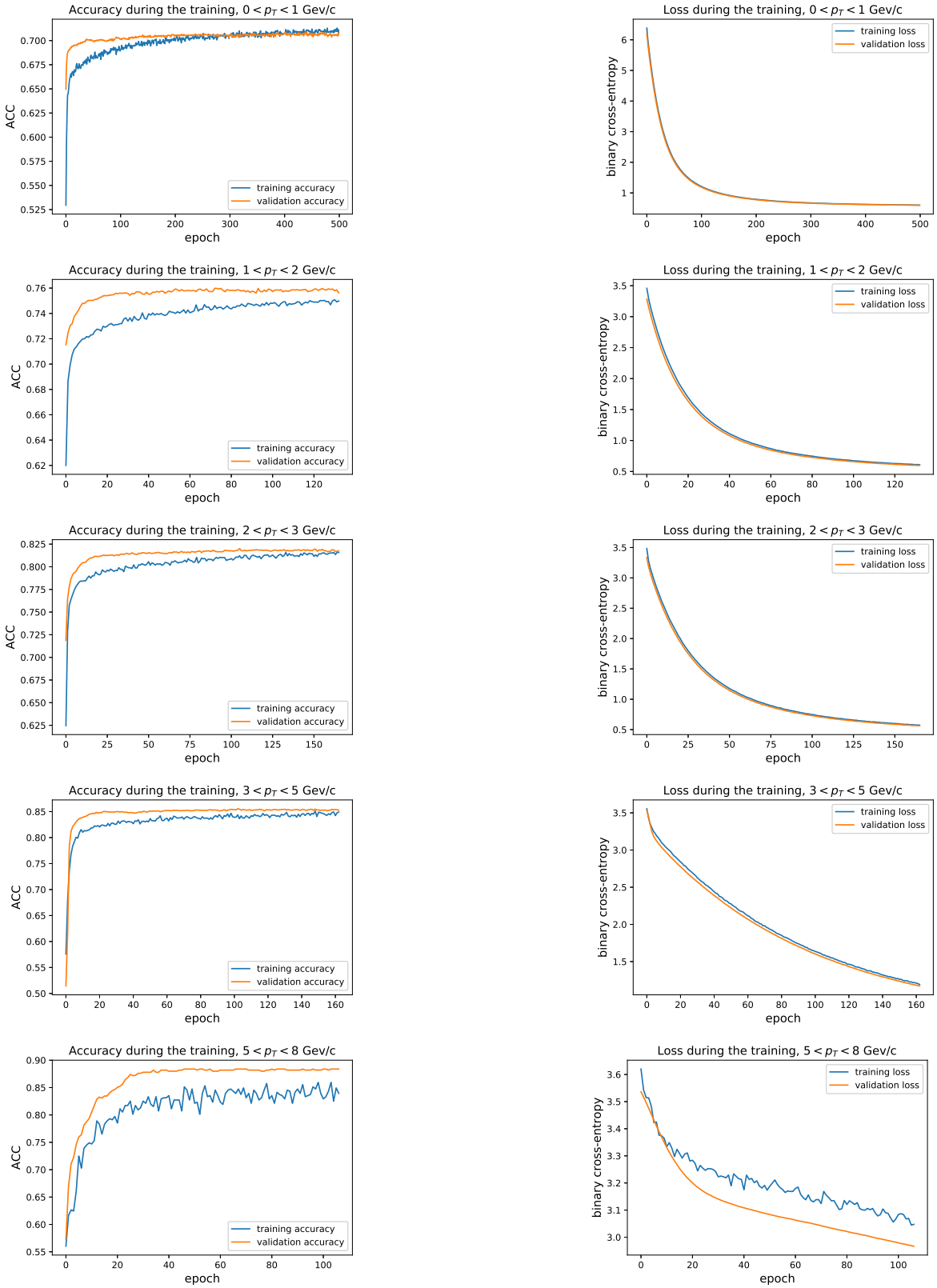


Figure 4.10: Accuracy (ACC) and binary cross-entropy vs epoch number for DNN model in the training stage, pre-cuts applied.

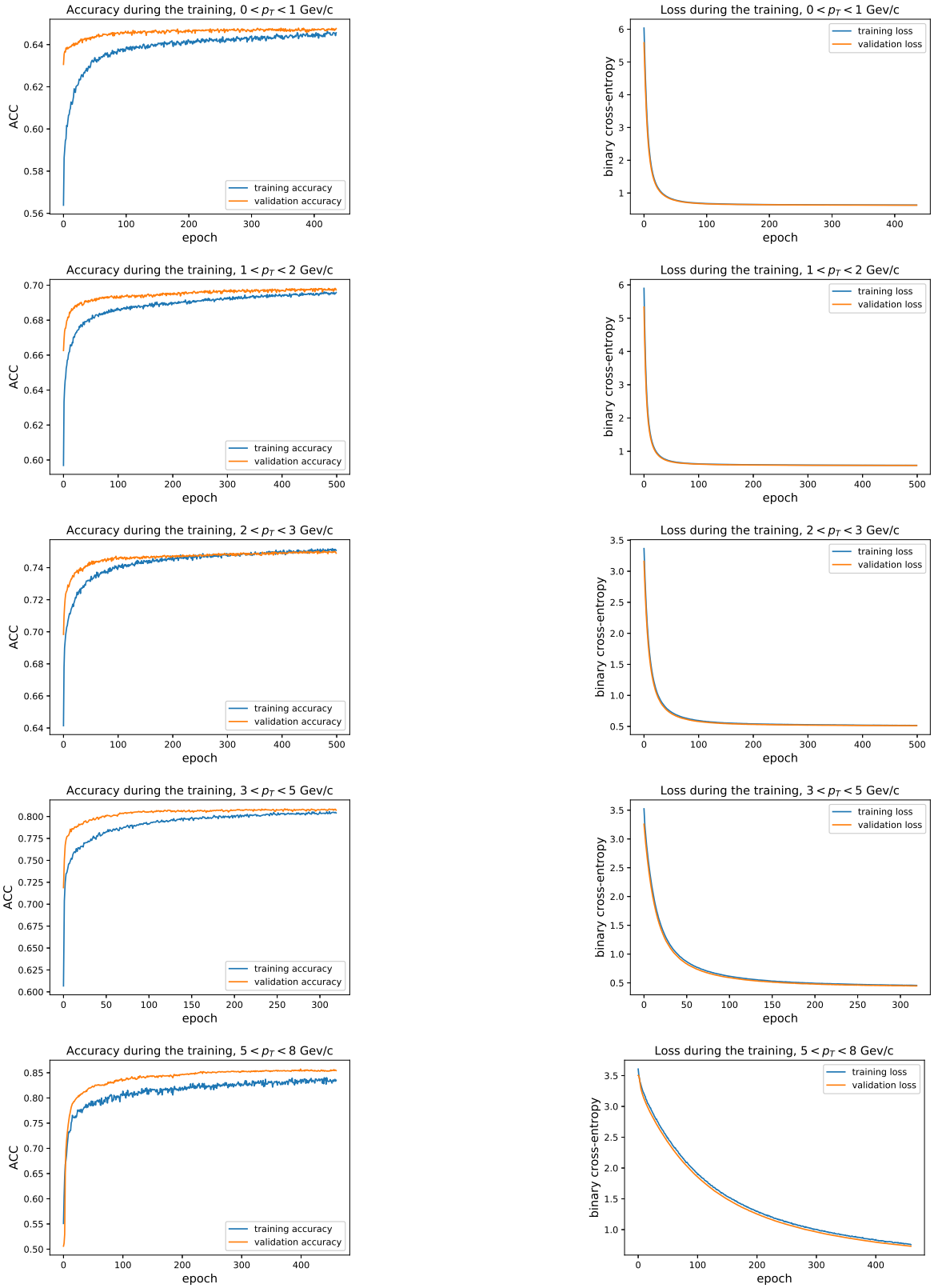
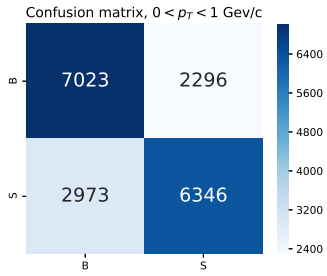
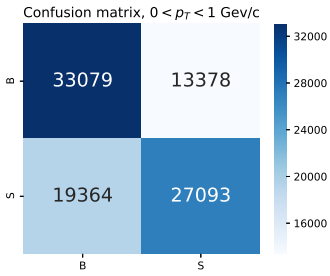


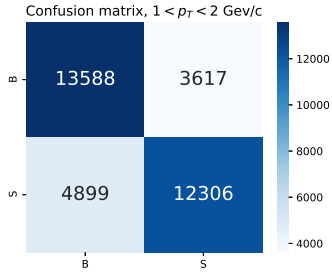
Figure 4.11: Accuracy (ACC) and binary cross-entropy vs epoch number for DNN model in the training stage, pre-cuts not applied.



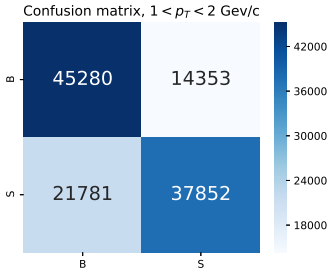
(a) p_{T} bin 0–1 GeV/c, pre-cuts applied.



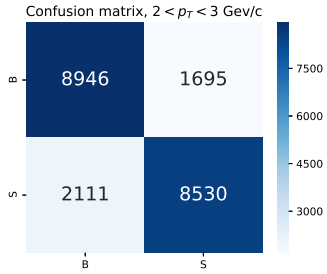
(b) p_{T} bin 0–1 GeV/c, pre-cuts not applied.



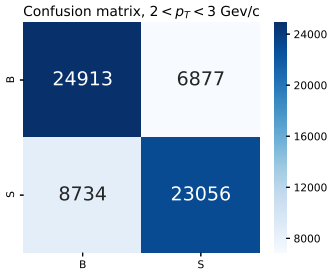
(c) p_{T} bin 1–2 GeV/c, pre-cuts applied.



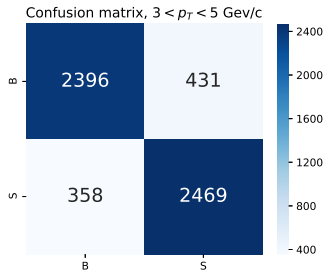
(d) p_{T} bin 1–2 GeV/c, pre-cuts not applied.



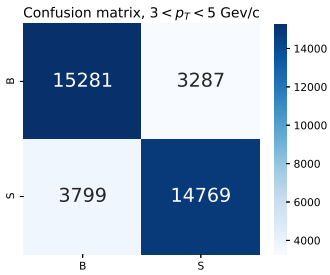
(e) p_{T} bin 2–3 GeV/c, pre-cuts applied.



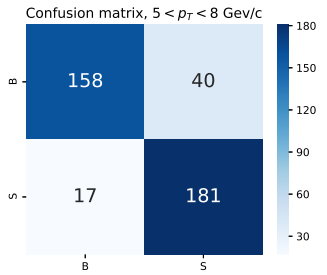
(f) p_{T} bin 2–3 GeV/c, pre-cuts not applied.



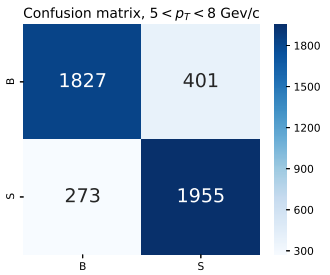
(g) p_{T} bin 3–5 GeV/c, pre-cuts applied.



(h) p_{T} bin 3–5 GeV/c, pre-cuts not applied.

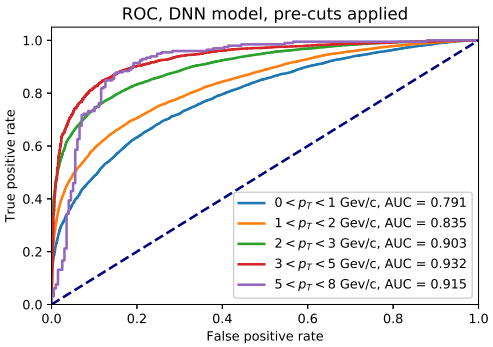


(i) p_{T} bin 5–8 GeV/c, pre-cuts applied.

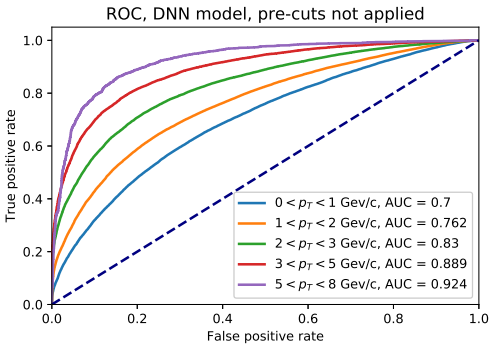


(j) p_{T} bin 5–8 GeV/c, pre-cuts not applied.

Figure 4.12: DNN model, comparison of confusion matrices for scenario with and without pre-cut application, evaluated over test set.



(a) ROC, pre-cuts applied.



(b) ROC, pre-cuts not applied.

Figure 4.13: DNN model, comparison of ROC and *AUC* for scenario with and without cut application, evaluated over test set.

4.1.3 Comparison of RF and DNN with boosted decision trees (BDT)

To compare the quality of optimized RF and DNN with BDT implemented in ROOT using TMVA library, similar analysis was performed by Ing. L. Kramárik from the Dep. of Physics, FNSPE. His optimized models with respect to *AUC* were used as a benchmark for comparison. Comparison of *AUC* values evaluated over test set for each p_{T} bin separately is available in Tables 4.11 (pre-cuts applied) and 4.12 (pre-cuts not applied). RF models seem to be the best approach in terms of *AUC* values when cuts are applied. DNN models were comparable with the qualities of BDT over the test set under pre-cuts. However, optimization of DNN models was the most time-consuming of all models presented in this thesis. Classifiers trained over the data under no pre-cut criteria achieved lower *AUC* values as expected. In the first two p_{T} bins (0–1 and 1–2 GeV/c), the best result was achieved by RF. In the rest of the p_{T} bins, RF performance was comparable to BDT performance. DNN model was comparable to BDT performance in all p_{T} bins except p_{T} bin 5–8 GeV/c, where achieved *AUC* was the lowest compared to other classifiers.

p_{T} bin id	p_{T} [GeV/c]	BDT *AUC*	DNN *AUC*	RF *AUC*
bin0	0–1	0.78	0.79	0.81
bin1	1–2	0.84	0.83	0.87
bin2	2–3	0.90	0.90	0.94
bin3	3–5	0.93	0.93	0.95
bin4	5–8	0.93	0.91	0.96

Table 4.11: Comparison of optimize classifiers over test set, pre-cuts applied.

p_{T} bin id	p_{T} [GeV/c]	BDT *AUC*	DNN *AUC*	RF *AUC*
bin0	0–1	0.70	0.70	0.73
bin1	1–2	0.76	0.76	0.79
bin2	2–3	0.83	0.83	0.82
bin3	3–5	0.89	0.89	0.90
bin4	5–8	0.94	0.92	0.94

Table 4.12: Comparison of optimized classifiers over test set, pre-cuts not applied.

4.1.4 Application of the existing classifier

As proposed in Sec. 1.1, application of the existing deep neural network classifier, trained over PYTHIA and measured data in [2] is to be tested. Since classifiers in [2] were trained only for p_{T} bins 1–2 GeV/c, 2–3 GeV/c, 3–5 GeV/c, test sets of matching p_{T} bins of HIJING simulation were used. First, feature homogeneity was tested for the signal and background representation used in [2] and in this thesis. The similarity of distributions was tested using Kolmogorov-Smirnov tests. As described in Fig. 4.4, 4.5, and 4.6, statistics wise, the homogeneity hypotheses were not always accepted on the significance level 1 %. However, as demonstrated by the application of the existing classifier to the HIJING data, similarity was sufficient machine-learning wise. The accuracy over balanced HIJING test set achieved similar rates as for test set used in [2]. Feature-wise standardization fitted and used for training set in [2] was applied per each p_{T} bin of the HIJING separately before the classification. In both cases, pre-processing using the same pre-cuts from Table 1.1 was applied. The absolute difference of *ACC* value evaluated over HIJING test set of classifier trained in [2] over PYHTIA signal and like-sign data background representation and the classifier trained over HIJING as well in this thesis was $\sim$ 3 % at maximum. The performance details of the classifier trained in [2] applied to the HIJING test set are presented in the Table 4.13 and by the confusion matrices in the Fig. 4.14.

p_{T} bin id	p_{T} [GeV/c]	*P*	*R*	*NPV*	*TNR*	*ACC*
bin1	1–2	0.83	0.60	0.69	0.87	0.74
bin2	2–3	0.86	0.70	0.75	0.88	0.79
bin3	3–5	0.89	0.75	0.78	0.91	0.83

Table 4.13: Evaluation of existing deep neural network classifiers over HIJING test set, pre-cuts applied.

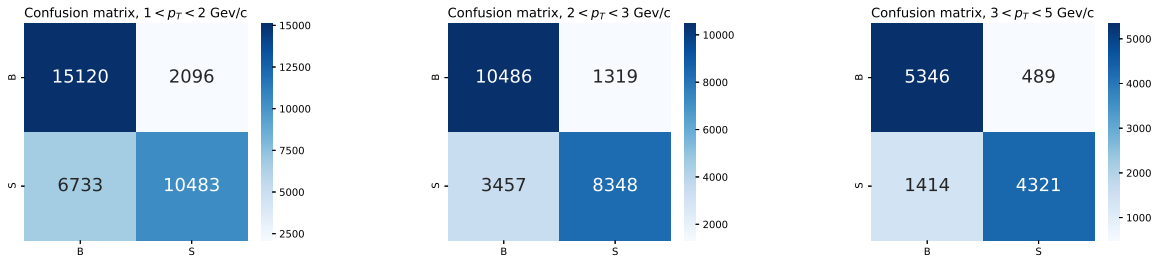


Figure 4.14: Confusion matrices, evaluation of existing deep neural network classifiers over HIJING test set, pre-cuts applied.

4.2 Data on QCD phase transitions

As proposed in Sec. 1.2, classification of equation of state (EoS) type based on the discrete representation (histogram 24×24 bins) of the joint distribution (p_T, Φ) of hadrons produced in the event was to be presented in this section. Data were simulated using a hybrid hydrodynamic model [4]. This model simulated the complete evolution of Au+Au nuclei collisions at $\sqrt{s_{NN}} = 200$ GeV and $\sqrt{s_{NN}} = 27$ GeV (energy per nucleon-nucleon pair). On the output, momenta and identities of hadrons were generated as if they came from a real collision. Two versions of the model were used. One included first order phase transition from hadronic matter to deconfined matter. The other version of the model assumed a smooth crossover between the two phases.

At the beginning, histogram inputs needed to be formed directly from available .root file or after rotating each event within transverse plane (p_x, p_y) using random angle $\alpha \sim U(-\pi, \pi)$. When the events are generated, the geometry is always such, that the two nuclei are shifted with respect to each other in the x-direction. This cannot be the case in real collisions. Therefore, in order to mimic the analysis in real data, momenta from the simulated events are randomly rotated in the transverse plane. When the rotation in the transverse plane was applied, we denote such scenario as event rotation of type "random". Otherwise we say, that no event rotation was applied.

The events used in the experiment were either of narrow centrality class (0–1 %, 20–21 % and 40–41 %) or wide centrality class (0–40 %). The effect of hadronic rescatterings in the simulation was tested as well by switching the rescattering parameter on and off for different sets of data. In accordance with [3] and to supress fluctuations, averaging of N histograms within one centrality class and one EoS type was also tested for $N \in \{2, 12, 22, 32, 42\}$. We have tested two averaging setups:

- random: N samples of histograms from given centrality class and EoS type were randomly selected and averaged to form single sample of the final input,
- batch: N samples of histograms from given centrality class and EoS type were randomly selected from subset of events coming from the same hydrodynamic evolution on top of which different sets of hadrons were Monte-Carlo generated. Next, they were again averaged to form one sample of final input.

For wide centrality class, the events of the same hydrodynamic evolution were stored in batches of 100 consecutive events in the provided .root file. We are aware that such averages can not be formed in the physics experiment, since information about event's EoS type would not be theoretically available. Nevertheless, the goal of this work was to test, if the information about EoS type was carried in event's (p_T, Φ) joint distribution and which conditions may affect it. For the purposes of the binary classification metrics evaluation, first order EoS type is considered as negative class C_0, crossover EoS type is considered as positive class C_1. Examples of QCD model's inputs that were fed to the models are presented in Fig. 4.15. All the inputs were in form of matrices created by histogram averaging and transformed with feature-wise standardization. Hence, the interpretation of the input is not straightforward in terms of quantities and units. Therefore, Fig. 4.15 presents examples of the inputs set to the original p_T and Φ directions only for easier interpretation.

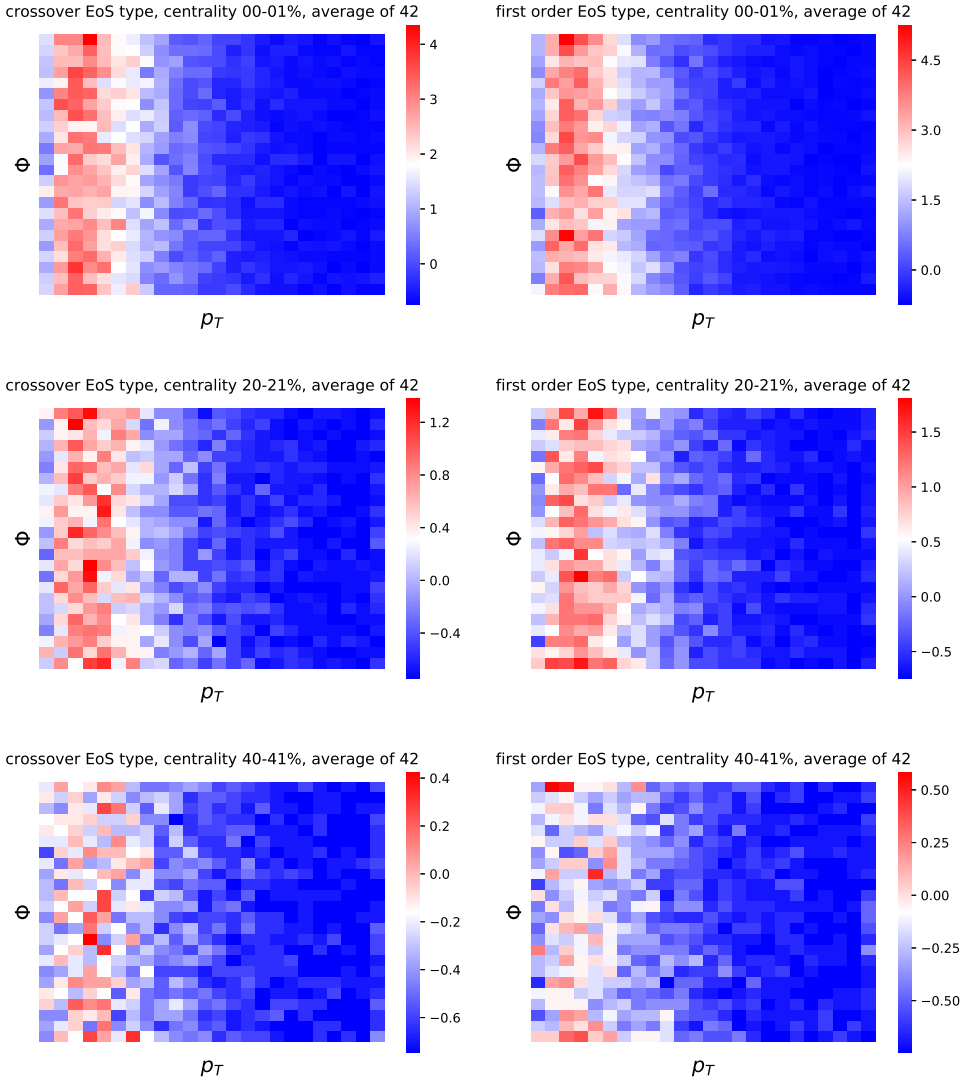


Figure 4.15: Examples of QCD model's inputs for N =42 after feature-wise standardization, $\sqrt{s_{\mathrm{NN}}}$=27 GeV.

4.2.1 Convolutional neural networks

As a benchmark architecture, similar convolutional neural netowork model as proposed in [3] was used as shown in Table 4.14. Although the loss function used in [3] is categorical crossentropy and output layer consists of two neurons with softmax activation, technically, it is the same as using the l_{BCE} for one output neuron activated with sigmoid function. In total, the model contained more than 200000 trainable parameters. We consider this model to be already sufficiently optimized with respect to ACC. All dropout levels are set to 0.2 except the last dropout rate, which was set to 0.5.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 24, 24, 16)	1040
batch_normalization_1 (Batch Normalization)	(None, 24, 24, 16)	64
p_re_lu_1 (PReLU)	(None, 24, 24, 16)	9216
dropout (Dropout)	(None, 24, 24, 16)	0
conv2d_2 (Conv2D)	(None, 24, 24, 16)	12560
batch_normalization_2 (Batch Normalization)	(None, 24, 24, 16)	64
p_re_lu_2 (PReLU)	(None, 24, 24, 16)	9216
dropout_1 (Dropout)	(None, 24, 24, 16)	0
average_pooling2d (Average Pooling)	(None, 12, 12, 16)	0
conv2d_3 (Conv2D)	(None, 12, 12, 32)	18464
p_re_lu_3 (PReLU)	(None, 12, 12, 32)	4608
dropout_2 (Dropout)	(None, 12, 12, 32)	0
average_pooling2d_1 (Average Pooling)	(None, 6, 6, 32)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147584
batch_normalization_3 (Batch Normalization)	(None, 128)	512
p_re_lu_4 (PReLU)	(None, 128)	128
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258
Total parameters: 203714		
Trainable parameters: 203394		
Non-trainable parameters: 320		

Table 4.14: Benchmark CNN model scheme.

First, experiment A was performed in order to verify the results. It was performed for simulated data of Au+Au collisions with $\sqrt{s_{\mathrm{NN}}}$ = 200 GeV as proposed in [3]. Binary classification metrics (ACC, P and R) are listed in Table 4.15. In general, better classification quality was achieved for higher values of N as expected.

Next, in experiment B, dependence of $\sqrt{s_{\mathrm{NN}}}$ was tested by performing exactly the same setup, but using simulated data with $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV. The classification quality was not reduced with the $\sqrt{s_{\mathrm{NN}}}$ value decreased.

In experiment C, effect of random event rotation was tested for data with $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV. As demonstrated in Table 4.15, rotating the event did not result in reduced classification quality as well.

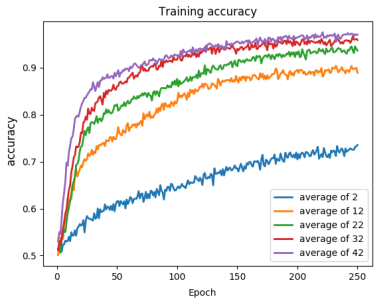
We did not detect any effect of hadronic rescatterings in the simulation during the experiment D, which tested classification qualities over $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV data with random event rotation. However, using narrow centrality types instead of wide centrality type as in experiments E and F seemed to have

major effect on the classification quality and speed of loss function minimization. Metrics available in Table 4.15 are evaluated over validation set (20 % of available data). The training of all experiments was performed over training set and unlike for D^0 experiment, no early stopping rule was applied since speed of overtraining was monitored as well. The training was stopped after exactly 250 epochs. In general, the higher the N value, the better was model's performance unless overtraining appeared, although generalization techniques were introduced heavily. In Fig. 4.16–4.21, training dynamics is visualized. Especially in 4.20, heavy overtraining was experienced (experiment E). However, during the initial epochs, some learning ability was present, therefore we assume that hyper-parameter optimization could lead to (at least small) learning dynamics improvement.

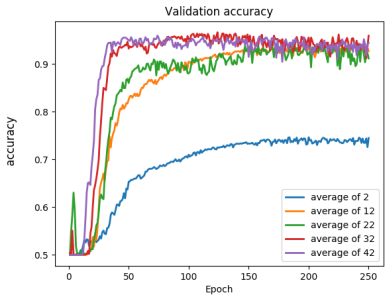
To test, if the convolutional layers focus on some physics-wise interpretable area in the inputs, feature maps for first order phase transition input type and crossover input type were visualized for each available convolutional layer (examples available in Fig. 4.22–4.25). Unfortunatelly, we were not able to draw evidence-based conclusion from the feature maps only. Since all the inputs fed to the benchmark model were transformed using feature-wise standardization fitted over training data, we assumed that the model focuses on gradient volume in p_{T} direction, since it is expected to be different for EoS types. To intentionally damage the inputs, feature-wise standardization was not fitted over the whole training set, but per class separately (and later for validation again applied per class). Hence, the gradient in p_{T} direction differences were reduced. The model was expected to reduce its performance and indeed it resulted in drastic reduction of classification ability (experiment G in Table 4.15). Training experiments A-B was performed over fixed training set size. The training set size was approximately 3000 samples (of averaged histograms = inputs) for experiments A and B, 16000 samples for experiments C-G, hence faster network convergence. The training set was balanced in terms of number of samples of both EoS classes.

Exp.	$\sqrt{s_{\mathrm{NN}}}$ [GeV]	event rotation	centrality	averaging setup	hadronic rescat.	N	*ACC %*	*P %*	*R %*
	200	none	narrow	random	on	2	74.47	77.35	69.17
	200	none	narrow	random	on	12	92.69	96.77	88.34
A	200	none	narrow	random	on	22	91.17	98.72	83.42
	200	none	narrow	random	on	32	95.94	99.72	92.14
	200	none	narrow	random	on	42	94.42	99.81	89.00
	27	none	narrow	random	on	2	81.26	79.70	84.15
	27	none	narrow	random	on	12	96.36	94.77	98.18
B	27	none	narrow	random	on	22	98.60	97.70	99.56
	27	none	narrow	random	on	32	99.40	99.69	99.10
	27	none	narrow	random	on	42	99.76	99.70	99.82
	27	random	narrow	random	on	2	80.39	78.40	84.17
	27	random	narrow	random	on	12	95.31	94.67	96.08
C	27	random	narrow	random	on	22	96.73	94.73	99.00
	27	random	narrow	random	on	32	97.63	96.77	98.85
	27	random	narrow	random	on	42	97.71	96.66	98.86
	27	random	narrow	random	off	2	80.77	78.82	84.16
	27	random	narrow	random	off	12	96.47	96.88	96.46
D	27	random	narrow	random	off	22	95.40	92.42	98.91
	27	random	narrow	random	off	32	97.41	95.93	99.03
	27	random	narrow	random	off	42	99.73	95.38	99.59
	27	random	wide	random	on	2	55.55	56.12	51.43
	27	random	wide	random	on	12	69.67	68.93	71.76
E	27	random	wide	random	on	22	71.88	65.49	92.67
	27	random	wide	random	on	32	69.02	62.03	98.23
	27	random	wide	random	on	42	50.56	50.31	99.99
	27	random	wide	batch	on	2	56.35	55.94	59.44
	27	random	wide	batch	on	12	60.33	57.73	77.48
F	27	random	wide	batch	on	22	61.15	57.83	82.65
	27	random	wide	batch	on	32	61.67	58.02	84.75
	27	random	wide	batch	on	42	65.06	63.72	70.14
	27	random	narrow	random	on	2	59.65	47,75	57.50
	27	random	narrow	random	on	12	59.32	75.66	28.09
G	27	random	narrow	random	on	22	59.72	59.77	60.72
	27	random	narrow	random	on	32	64.09	80.47	37.70
	27	random	narrow	random	on	42	69.01	75.92	56.15

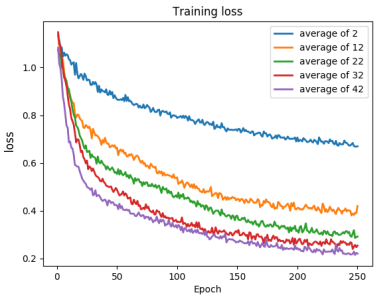
Table 4.15: Evaluation of the classifiers over validation set for different data pre-processing types



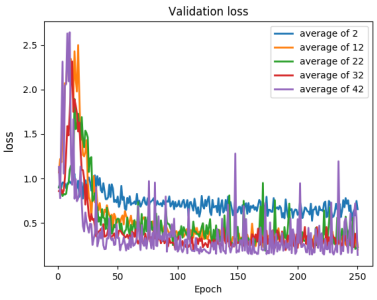
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.

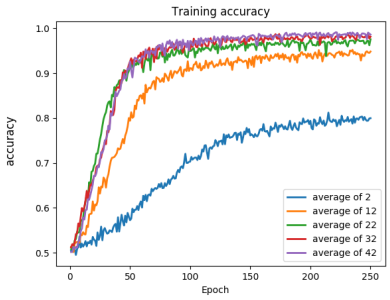


(c) Train loss during the training.

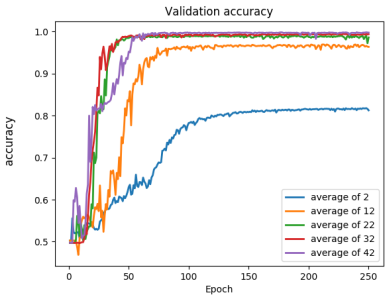


(d) Valid. loss during the training.

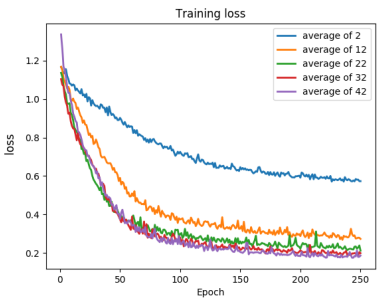
Figure 4.16: Experiment A: Training validation plots, $\sqrt{s_{\mathrm{NN}}} = 200$ GeV, no event rotation, narrow centrality class, random averaging setup, post hydro rescattering included.



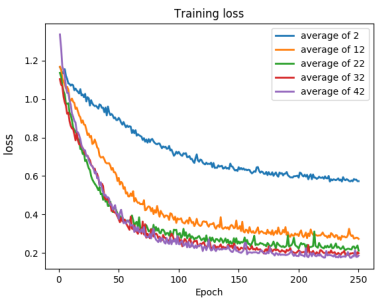
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.

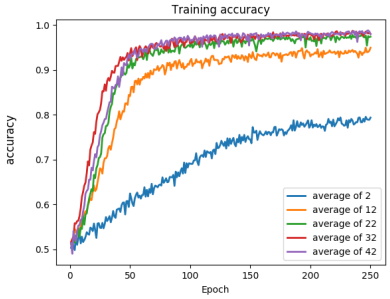


(c) Train loss during the training.

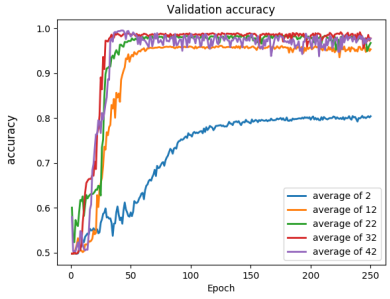


(d) Valid. loss during the training.

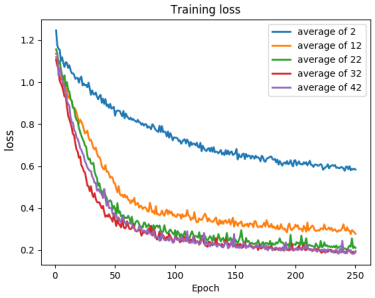
Figure 4.17: Experiment B: Training validation plots, $\sqrt{s_{\mathrm{NN}}} = 27$ GeV, no event rotation, narrow centrality class, random averaging setup, post hydro rescattering included.



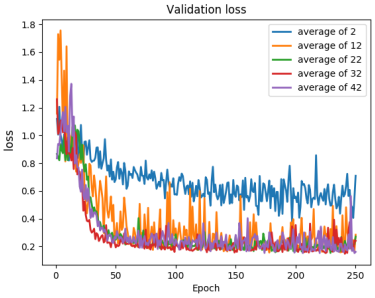
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.

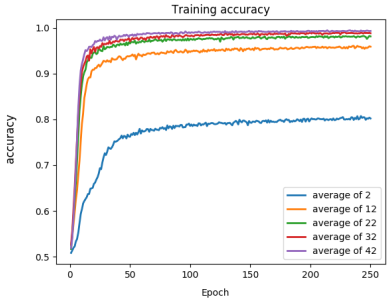


(c) Train loss during the training.

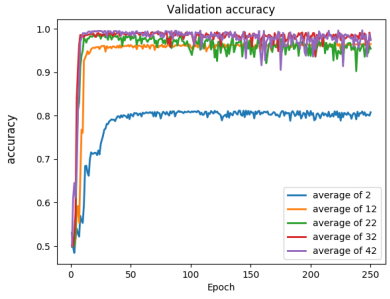


(d) Valid. loss during the training.

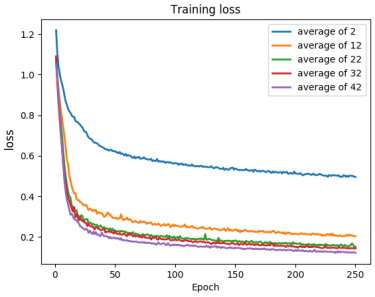
Figure 4.18: Experiment C: Training validation plots, $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV, random event rotation, narrow centrality class, random averaging setup, post hydro rescattering included.



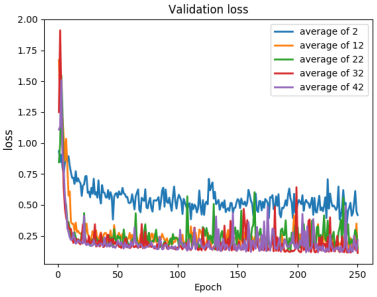
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.

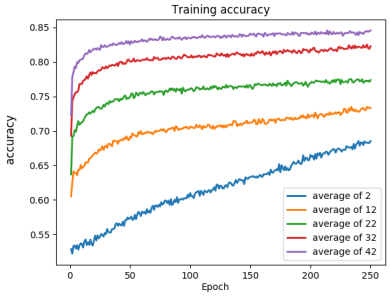


(c) Train loss during the training.

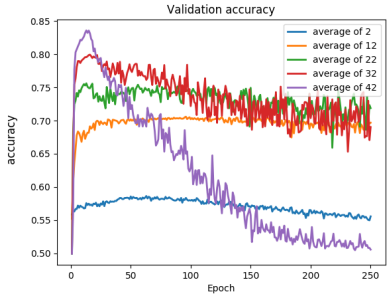


(d) Valid. loss during the training.

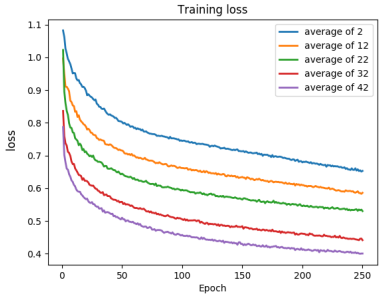
Figure 4.19: Experiment D: Training validation plots, $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV, random event rotation, narrow centrality class, random averaging setup, post hydro rescattering excluded.



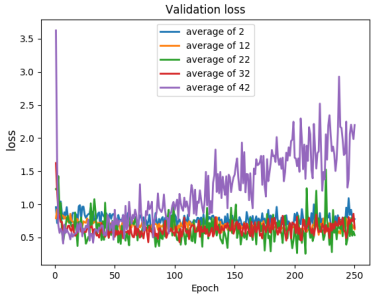
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.

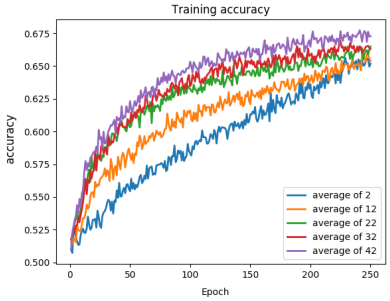


(c) Train loss during the training.

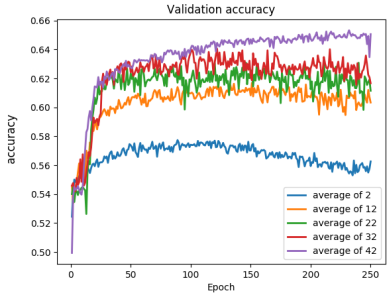


(d) Valid. loss during the training.

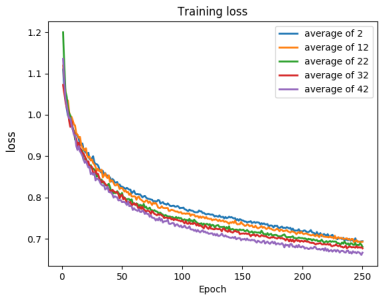
Figure 4.20: Experiment E: Training validation plots, $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV, random event rotation, wide centrality class, random averaging setup, post hydro rescattering included.



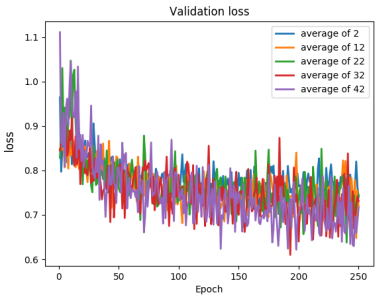
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.

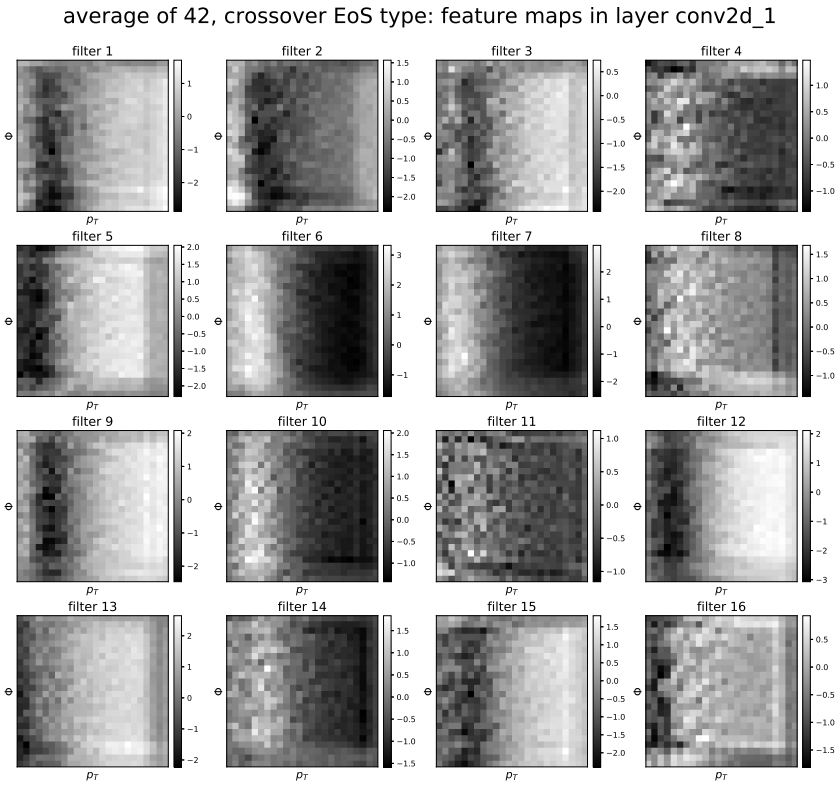


(c) Train loss during the training.

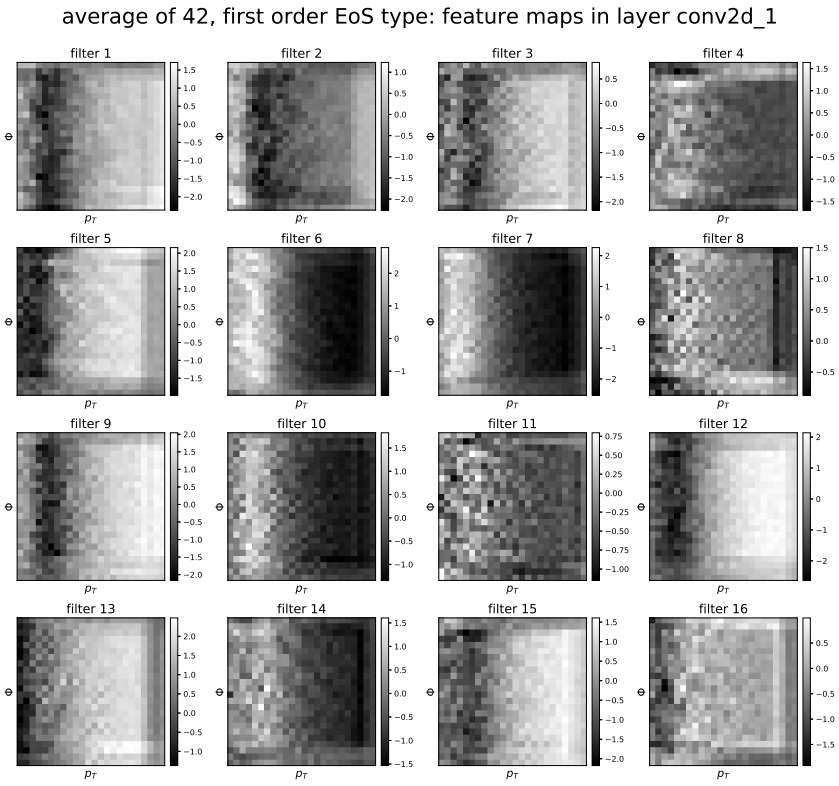


(d) Valid. loss during the training.

Figure 4.21: Experiment F: Training validation plots, $\sqrt{s_{\mathrm{NN}}}$ = 27 GeV, random event rotation, wide centrality class, averaging within bunches, post hydro rescattering included.

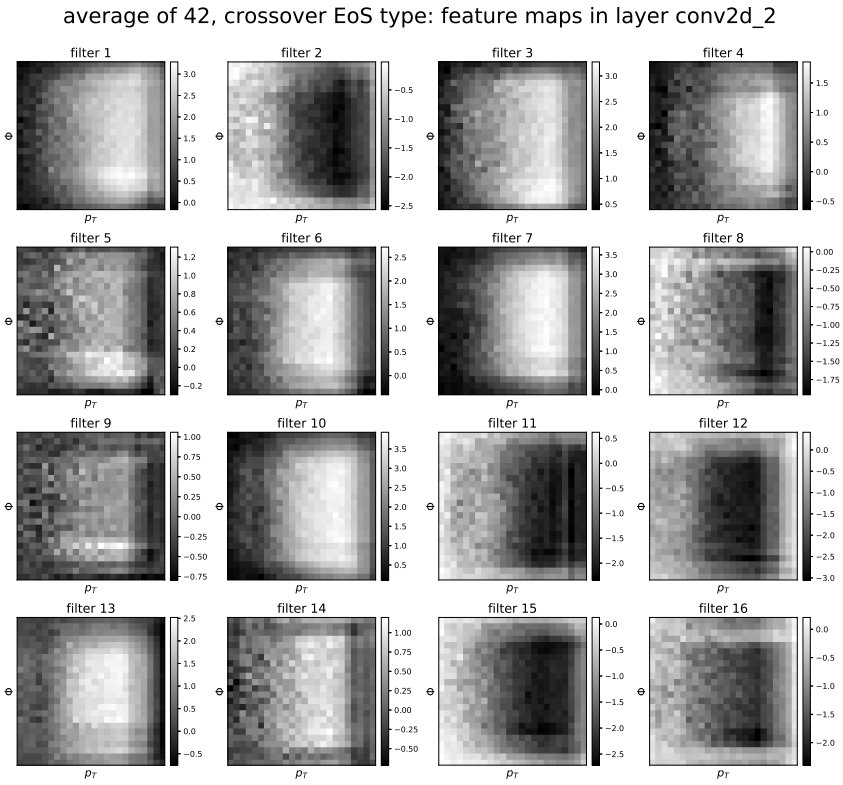


(a) Crossover EoS type sample.

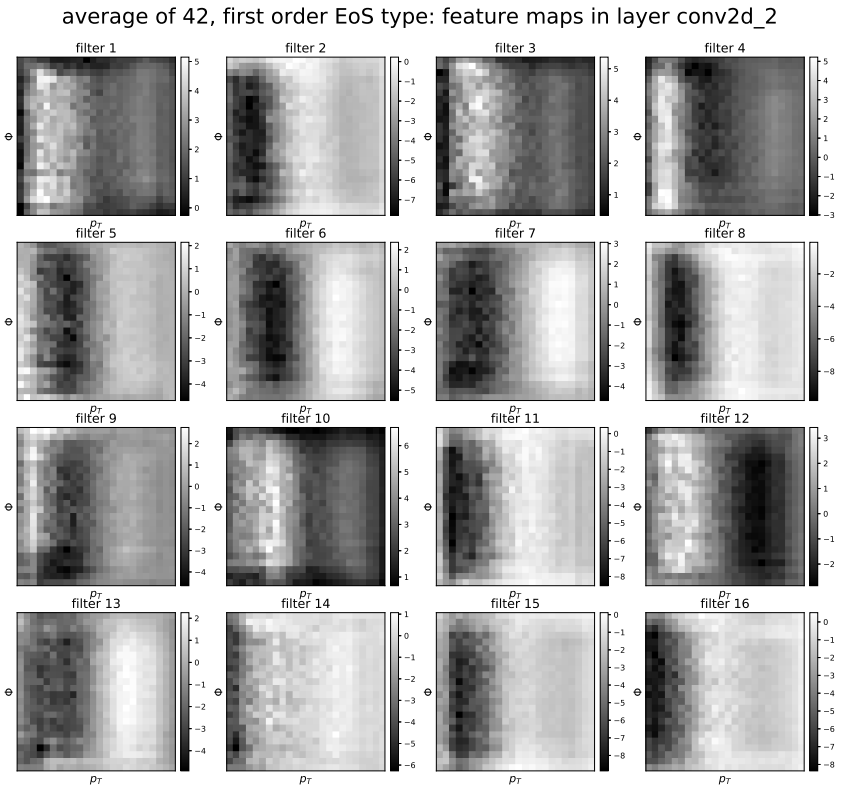


(b) First order EoS type sample.

Figure 4.22: Feature maps of first convolutional layer of architecture. Feature map shows transformed distribution of (Φ, p_{T}) using filter's optics.



(a) Crossover EoS type sample.



(b) First order EoS type sample.

Figure 4.23: Feature maps of second convolutional layer of architecture. Feature map shows transformed distribution of (Φ, p_{T}) using filter's optics.

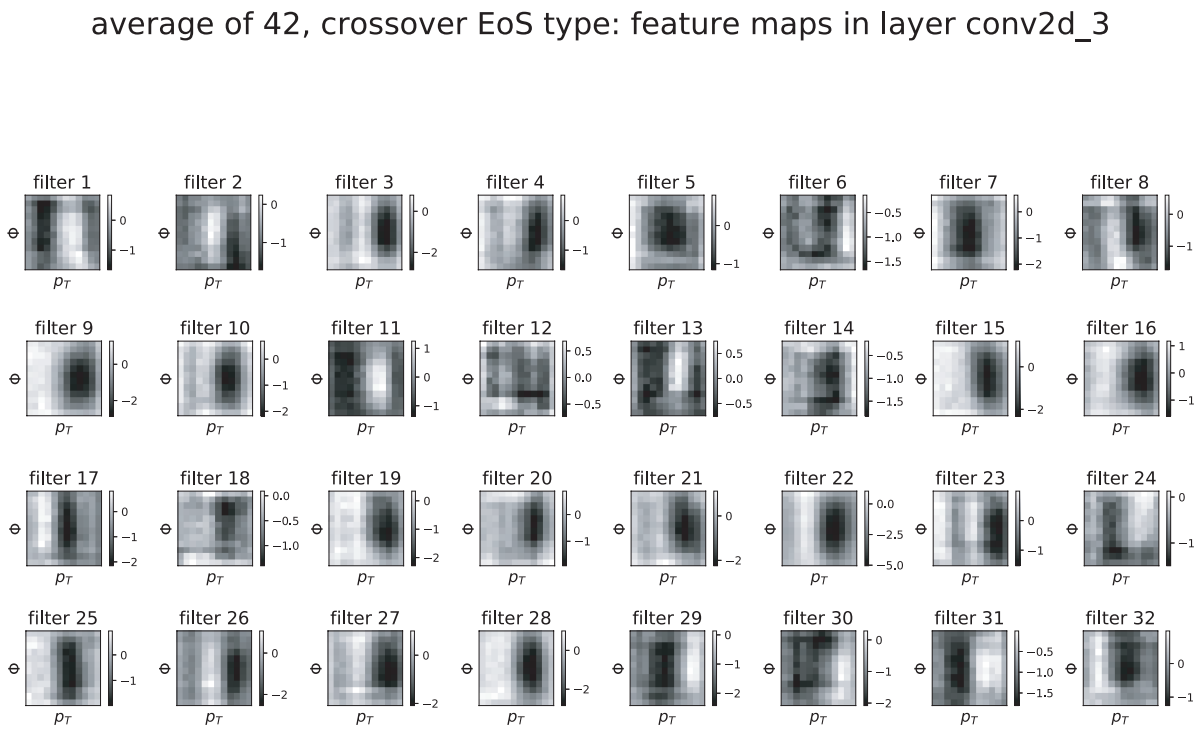


Figure 4.24: Feature maps of third convolutional layer of architecture, crossover EoS type sample. Feature map shows transformed distribution of (Φ, p_{T}) using filter's optics.

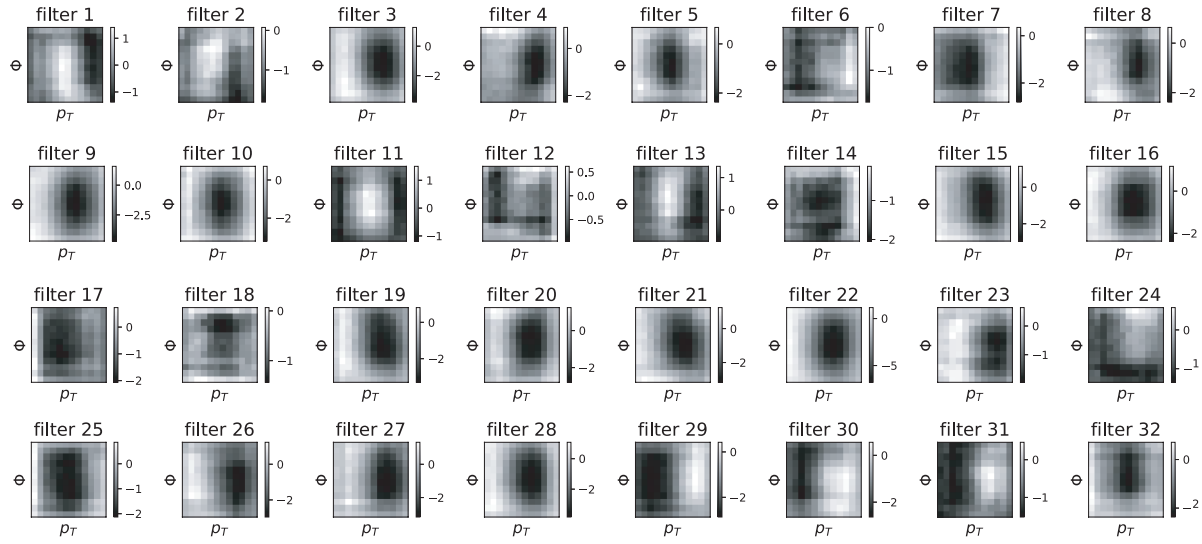


Figure 4.25: Feature maps of third convolutional layer of architecture, first order EoS type sample. Feature map shows transformed distribution of (Φ, p_{T}) using filter's optics.

4.2.2 Residual neural networks

Although adding residual blocks is suitable for deeper architectures than the benchmark model is, residual neural network model based on the benchmark model was built with more than 6 hundred thousands trainable parameters. The model's scheme is displayed in the Table 4.16. However, as described in Table 4.17, no significant improvement was achieved in terms of classification quality.

Layer (type)	Output Shape	Param #	Connected to
conv2d_56 (Conv2D)	(None, 24, 24, 16)	1040	input_17
batch_normalization_36 (Batch Norm.)	(None, 24, 24, 16)	64	conv2d_56
p_re_lu_53 (PReLU)	(None, 24, 24, 16)	9216	batch_normalization_36
dropout_59 (Dropout)	(None, 24, 24, 16)	0	p_re_lu_53
conv2d_57 (Conv2D)	(None, 24, 24, 16)	12560	dropout_59
add_17 (Add)	(None, 24, 24, 16)	0	dropout_59, conv2d_57
re_lu_7 (ReLU)	(None, 24, 24, 16)	0	add_17
conv2d_58 (Conv2D)	(None, 24, 24, 32)	18464	re_lu_7
p_re_lu_54 (PReLU)	(None, 24, 24, 32)	18432	conv2d_58
dropout_60 (Dropout)	(None, 24, 24, 32)	0	p_re_lu_54
average_pooling2d_19 (Average Pooling)	(None, 24, 24, 32)	0	dropout_60
flatten_18 (Flatten)	(None, 4608)	0	average_pooling2d_19
dense_35 (Dense)	(None, 128)	589952	flatten_18
batch_normalization_37 (Batch Norm.)	(None, 128)	512	dense_35
p_re_lu_55 (PReLU)	(None, 128)	128	batch_normalization_37
dropout_61 (Dropout)	(None, 128)	0	p_re_lu_55
dense_36 (Dense)	(None, 2)	258	dropout_61
Total parameters: 650626			
Trainable parameters: 650338			
Non-trainable parameters: 288			

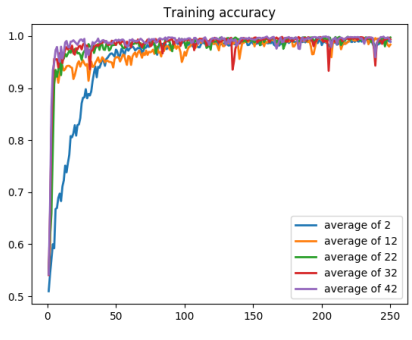
Table 4.16: Residual neural network model scheme

Exp.	$\sqrt{s_{\mathrm{NN}}}$ [GeV]	event rotation	centrality	averaging setup	hadronic rescat.	N	*ACC %*	*P %*	*R %*
	27	random	narrow	random	on	2	69.82	65.71	83.53
	27	random	narrow	random	on	12	94.88	96.40	93.30
R	27	random	narrow	random	on	22	96.72	98.11	95.30
	27	random	narrow	random	on	32	97.21	95.54	99.07
	27	random	narrow	random	on	42	81.50	73.10	99.99

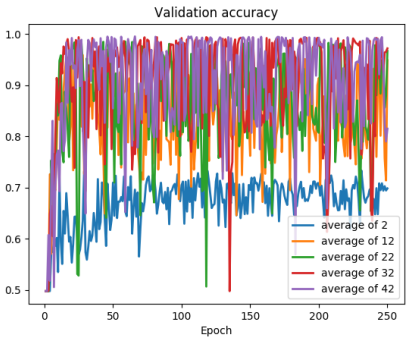
Table 4.17: Evaluation of the residual and benchmark classifier.

As visualized in Fig. 4.26, model's learning process suffered from instabilities and fluctuations, therefore the experiment R resulted in worse overall classification performance than experiment C in terms of binary classification metrics. The fluctuations were indeed caused by fast overtraining. However, they were present even before the phase of overtraining was achieved as visible in validation ac-

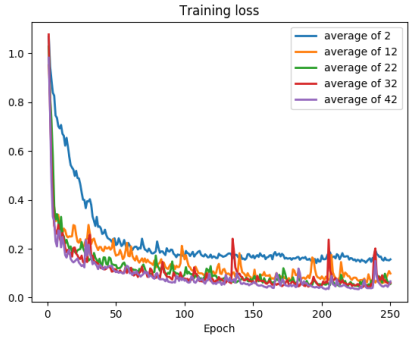
curacy development for $N = 2$. Although improving the benchmark model classification ability using residual model first seemed to be a reasonable next step, during the experimental phase a more important question appeared. Learnt classification ability during one experiment (e.g. experiment C) was not directly applicable to the input data of other experiment (e.g. experiment B) transformed with feature-wise standardization fitted during the experiment C. AUC of such classification achieved $\sim$ 50 %, therefore was comparable to the random classification. Such cross-classification problem was also present when applying the model from experiment C to the input data of experiment A. Further exploration of possible data standardization techniques to enable successful transfer learning inbetween the data from different types of the simulation is therefore necessary.



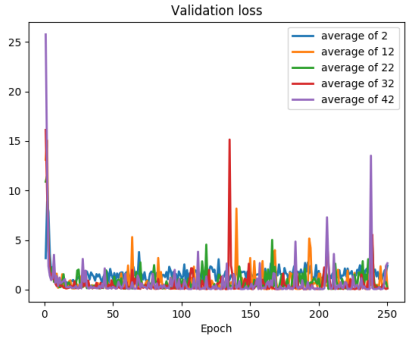
(a) Train accuracy during the training.



(b) Valid. accuracy during the training.



(c) Train loss during the training.



(d) Valid. loss during the training.

Figure 4.26: Experiment R: Training validation plots, $\sqrt{s_{\mathrm{NN}}} = 27$ GeV, random event rotation, narrow centrality, random averaging setup, post hydro rescattering included.

Conclusion

In this study we compared multiple machine learning approaches and data pre-processing dependencies for two heavy ion physics problems: D^0 decay classification and identification of the nature of the QCD phase transition.

For D^0 decay classification problem, HIJING simulation of d+Au collision with $\sqrt{s_{NN}} = 200$ GeV/c was provided by the STAR collaboration. We have performed homogeneity tests using Kolmogorov-Smirnov test of two possible background representations: non-signal unlike-sign pairs and like-sign pairs. The homogeinity was accepted on significance level 1 % for all tested distributions within all selected p_T bins. Therefore non-signal unlike-sign pairs were selected as background class representation. Two machine learning models were trained and optimized with respect to AUC over feature-wise standardized training set: random forest model (RF) and deep neural network model (DNN). The features used in classification were DCA_π, DCA_K, l_{decay}, $DCA_{\pi,K}$, $\cos\theta$, DCA_{D^0} and $\cos\theta^*$. Two different preprocessing approaches were tested as well. First approach included training the models over the data under following pre-cut criteria: $0.002 < DCA_\pi < 0.2$ cm, $0.002 < DCA_K < 0.2$ cm, $0.0005 < l_{decay} < 0.2$ cm, $DCA_{\pi,K} < 0.02$ cm, $\cos\theta > 0.7$ and $DCA_{D^0} < 0.05$ cm. The second approach applied no such preprocessing. The problem was partitioned into five dijoint p_T bins: 0–1 GeV/c, 1–2 GeV/c, 2–3 GeV/c, 3–5 GeV/c and 5–8 GeV/c. The models were trained over balanced data set and compared not only with respect to each other, but also with respect to optimized boosted decision tree model (BDT) implemented by Ing. L. Kramárik. Standard binary classification metrics were evaluated such as P, R, NPV, TNR and most importantly, AUC. Classifiers trained over the data under no pre-cut criteria achieved lower AUC values than classifiers trained over data under pre-cuts. The most successful classifier over test set under pre-cuts was RF achieving AUC values 81 %–96 % based on the p_T bin (higher p_T value resulted in better classification). For the test set were no pre-cuts were applied, BDT, RF and DNN models performance was comparable with respect to AUC value. However, for p_T bins 0–1 GeV/c and 1–2 GeV/c, RF achieved better performance. The AUC values for the scenario without pre-cuts ranged between 73 %–94 % for RF, 70 %–92 % for DNN and 70 %–94 % for BDT depending on p_T bin. The existing classifier trained in [2] over PYTHIA simulation and measured data was applied to the HIJING test set as well. The classification quality achieved $ACC = 74$ % in p_T bin 1–2 GeV/c, $ACC = 79$ % in p_T bin 2–3 GeV/c and $ACC = 83$ % in p_T bin 3–5 GeV/c, which is comparable to the values achived over test sets in [2].

For the identification of the nature of the QCD phase transition, data simulated using a hybrid hydrodynamic model [4] were provided by Ing. J. Cimerman. The model simulated the complete evolution of Au+Au nuclei collisions at different energies per nucleon-nucleon pair. Two versions of the model were used, one version assumed first order phase transition, the second version assumed a smooth crossover between the two phases. We trained convolutional neural network model (CNN) based on [3] and applied it to the data under different pre-processing conditions. Classification quality dependence on parameters of the simulation, such as centrality class, $\sqrt{s_{NN}}$ value and the effect of hadronic rescatterings was tested as well. The classification was performed based on the discrete representation (histogram) of the joint distribution (p_T, Φ) of hadrons produced in the event. The histograms were produced directly from the .root files produced by the simulation, or after each event was rotated randomly for random angle $\alpha \sim U(-\pi, \pi)$ within transverse plane (p_x, p_y). To supress fluctuations, N histograms within one centrality class and one EoS type were averaged into one input. Tested values of N were 2, 12, 22, 32 and 42.

Based on the experiments we performed, we did not detect reduced quality of the classifier when $\sqrt{s_{NN}}$ value decreased from 200 GeV to 27 GeV. The classification was also not influenced by the presence of hadronic rescatterings in the simulation or by the random rotation of the events within transverse plane. The classifier's *ACC* over test set ranged from 74.47 % to 99.73 % based on the N value (the higher the N, the better was classification quality). However, we detected strong effect of centrality class selection. When centrality class was changed from narrow type (0–1 %, 20–21 %, 40–41 %) to wide type (0–40 %), the classification quality dropped significantly (*ACC* ranged from 55 % to 65 %). We have tested the classifier's dependence on the gradient volume in p_{T} direction by intentionally removing this feature by applying feature-wise standardization fitted over each EoS type separately. The model's performance indeed dropped. *ACC* ranged from 60 % to 69 % for the narrow centrality class. In order to improve the learning of the experiments with standard application of feature-wise standardization, residual block was added to the CNN architecture. However, such approach resulted in instability during the training and lowered values of binary classification metrics. We explored the possibility of direct application of the model fitted over one set of simulation and pre-processing parameters to the data pre-processed or simulated differently. Such classification achieved $AUC \sim 50$ %, therefore was comparable to the random classification. Further exploration of possible data standardization techniques to enable successful transfer learning inbetween the data from different types of the simulation is therefore necessary in order to understand the differences between high-level features as seen by the CNN models.

Bibliography

[1] K. Hladká: *Statistics and machine learning applied to heavy nuclei collision products*. Bachelor's Project, FNSPE CTU Prague, 2019.

[2] K. Hladká: *Optimization of classification techniques for heavy nuclei collision products*. Research Project, FNSPE CTU Prague, 2020.

[3] Y. Du, K. Zhou, J. Steinheimer, L. Pang, A. Motornenko, H. Zong, X. Wang, H. Stöcker: *Identifying the nature of the QCD transition in relativistic collision of heavy nuclei with deep learning*. The European Physical Journal C 80, 2020, 516:1-17.

[4] Iu. A Karpenko, P. Huovinen, H. Petersen, M. Bleicher: *Estimation of the shear viscosity at finite net-baryon density from A+A collisions data at* $\sqrt{s_{NN}}$ *= 7.7–200 GeV.* Physical Review C 91, 2015, 064901.

[5] K. He, X. Zhang, S. Ren, J. Sun: *In '2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)'*. IEEE Xplore, 2016, 770-778.

[6] C. C. Aggarwal: *Neural Networks and Deep Learning*. Springer International Publishing, 2018.

[7] C. M. Bishop: *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[8] B. Zitová, A. Novozámský: *Rozpoznávání obrazu 1*. Lectures, FNSPE CTU Prague, 2020.

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

[10] D. Tlusý: *A Study of Open Charm Production in p+p Collisions at STAR*. PhD Dissertation, FNSPE CTU Prague, 2014.

[11] G. Louppe: *Understanding random forests from theory to practise*. PhD Dissertation, University of Liège, Faculty of Applied Sciences, Department of Electrical Engineering and Computer Science, 2014.

[12] K. He, X. Zhang, S. Ren, J. Sun: *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. arXiv:1502.01852, 2015.

[13] L. Brieman: *Random Forests*. Statistics Department, University of California, 2001.

[14] L. Pardo: *Statistical inference based on divergence measures*. Chapman & Hall/CRC, 2006.

[15] P. Bouř: *Development of statistical nonparametric and divergence methods for data processing in D0 and NOvA experiments*. Master's Thesis, FNSPE CTU Prague, 2016.

[16] T. Kohonen *Self-organized formation of topologically correct feature maps*. Biological Cybernetics 43, 1982, 59–69.

[17] T. Kohonen, T. Honkela: *Kohonen network*. Scholarpedia, peer-reviewed open-access encyclopedia, 2007, 2(1):1568. `http://www.scholarpedia.org/article/Kohonen_network`

[18] F. Pedregosa et al.: *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research 12, 2011, 2825-2830.

[19] F. Chollet et al.: *Keras*. 2015. `https://keras.io`

[20] A. LeNail: *NN-SVG: Publication-ready NN-architecture schematics.*. Journal of Open Source Software, 4(33), 2019, 747.