Tracks and showers classification with Graph Neural Networks for the KM3NeT/ARCA detector

Francesco Carenini
Exam Applied Machine Learning: Basics

October 2, 2023

Abstract

In this document, I present a track-shower classification developed with Graph Neural Networks and applied to the Monte Carlo simulations of the KM3NeT/ARCA detector in a 21 strings configuration. This classifier is currently being developed for the online framework of the ARCA detector, responsible for the real-time analysis of incoming events in the detector, possibly linking them to the observation of other messengers (gamma-rays, gravitational waves and neutrinos) performed by other instruments. However, the novelty of this study lies in the classification of tracks and showers using a new training dataset which is only constituted by the simulation of neutrino interactions in the seawater (both CC and NC), and not on simulation packages specifically designed for bundles of atmospheric muons, which leave a clear track-like signature.

Contents

1 Neutrino detection with KM3NeT/ARCA

KM3NeT/ARCA, located 100 km off-shore Portopalo di Capo Passero in Sicily, is dedicated to high-energy neutrino studies (up to multi-PeV). Once completed, it will have cubic kilometers of volume sensitive to the detection of neutrino interactions. It will consist of two 'building blocks', for a total of 230 Detection Units (DUs). Each DU carries 18 digital optical modules, with 31 photomultiplier tubes (PMTs), detecting Cherenkov radiation induced by charged particles, produced by the interaction of neutrinos in the seawater. Muons generated in charged current interactions travel trough the volume of the detector and leave a track-like signature. KM3NeT reconstruction algorithms are able to infer the energy and direction of the incoming neutrinos, starting from PMT information. The dataset used in this analysis is referred to Monte Carlo (MC) productions considering ARCA in a 21 strings configuration (ARCA21).

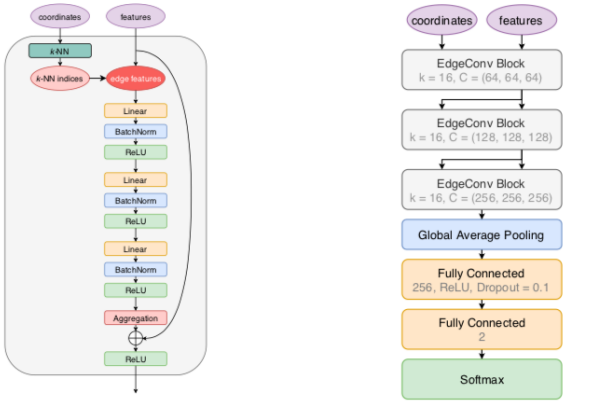
1.1 Event signatures in neutrino telescopes

Two classes of events can be distinguished in a neutrino telescope: tracks and showers (also called cascades). Track-like events are mostly generated by charged current (CC) interactions of ν_μ (or $\bar{\nu}_\mu$) where a muon appears in the final state, inducing Cherenkov light and leaving a well-defined track, while shower-like events do not have a clear signature. The golden channel of neutrino telescopes is represented by the CC interaction of a muon neutrino, which produces a hadronic shower and a muon. However at energies above the TeV level, the length of the muon signature in water is 3 times larger with respect to hadronic shower development. So, in such a situation, only the muon track is clearly seen. CC interaction of ν_e produces both hadronic and electromagnetic showers. The resulting high-energy electron has a high probability to radiate a photon via bremsstrahlung after few tens of cm of water/ice, then pair production happens and the electromagnetic cascade develops, until particles involved are below the critical energy. Neutral Current (NC) scattering of neutrinos produces a hadronic shower. Because of event-by-event fluctuations of hadronic cascades, this class of events cannot be distinguished from ν_e CC signature.

2 Machine learning with KM3NeT/ARCA

A KM3NeT event is constituted by hits and each hit has its own coordinates, namely *x, y, z*, alongside the time t of the detection and the pointing direction of the PMT that recorded the hit. This is the reason why Graph Neural Networks (GNNs) are quite suitable to perform deep learning classifications. Indeed, the model relies on the fact that each hit is a node and the features of the nodes are the previously mentioned coordinates of that particular hit. In addition to that, GNNs do not need rigid pre-chosen spatial and temporal bins to model the detector, making them very suitable for a moving detector as KM3NeT/ARCA. Finally, even if MC simulations still contain the reconstruction steps of each event, both as track and shower, those features are no needed as inputs to the GNNs, since they only rely on the raw information of the event. The GNN model implemented in KM3NeT is based on the ParticleNet architecture (see Figure 1), which makes extensive use of EdgeConv

(edge convolution block) operations and adopts the dynamic graph update approach. This



(a) Structure of an EdgeConv block.

(b) Architecture of ParticleNet.

Figure 1: The structure of ParticleNet and the EdgeConv block. coordinates = [**pos**, time], features = [**pos, dir**, time].

block was adapted for jet tagging at LHC, but can be also used for high dimensional and sparse signals, since they resemble a point cloud [1]. The Edge Convolutional block has two separate inputs: the coordinates include both position and time information, they are used to determine which nodes are connected to the others; the second input is seven dimensional and it is used to calculate the features, it contains also information about the direction of the PMT. Each node is connected to its k nearest neighbours based on the distance of the coordinates: coordinate input is supplied with x,y,z,ct, beeing c the speed of light in water, and the difference of these coordinates between nodes is used for the calculation of the neighbours. For this work, the euclidean distance is used: $d^2 = c^2\delta t^2 + \delta x^2 + \delta y^2 + \delta z^2$. Then, k is an hyper-parameter of the model and will be kept fixed at $k = 16$. EdgeConv starts by representing an event as a graph, whose vertices are the hits on the PMTs, and the edges are constructed as connections between each hit to its k nearest neighboring hits. The EdgeConv operation for each node x_i has the form:

$$x_i^{'} = \square_{j=1}^{k}\ h_\Theta(\mathbf{x}_i, \mathbf{x}_{ij}),$$

where the square represents the symmetric aggregation operator (a mean in this case) and h_Θ is the edge function. In particular, $h_\Theta(\mathbf{x}_i, \mathbf{x}_{ij}) = \bar{h}_\Theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$: it is implemented as a three-layer MultiLayer Perceptron (MLP) whose parameters are shared among the edges. In addition, one of the properties of Edge Convolution is to provide a dynamic graph, such that the connections in the graph are re-calculated based on the output of the previous block. The feature vectors learned by EdgeConv can be viewed as new coordinates of the original points. After the third EdgeConv block, a global average pooling is performed, reshaping the nodes features, and finally two dense layers are added, producing the output of the network.

3 Dataset

The goal of this analysis is to classify tracks and showers with GNNs, working with ARCA21. Because of that, the dataset comes from the run-by-run MC production (version 8.1) for

ARCA21 (supervised learning). MC simulations are saved in root files and stored in Lyon CC. The training sample of the GNNs consists of $\sim$ 400.000 events, where 50% of them belong to the class of $\overset{(-)}{\nu}_\mu$ CC interactions, while the remaining 50% is shared among $\overset{(-)}{\nu}_e$ CC, $\overset{(-)}{\nu}_e$ NC and $\overset{(-)}{\nu}_\mu$ NC. In particular, since electromagnetic showers are better resolved in a neutrino telescope with respect to hadronic ones, the 50% of shower-like events will be divided into: 30% $\overset{(-)}{\nu}_e$ CC, 10% $\overset{(-)}{\nu}_e$ NC and 10% $\overset{(-)}{\nu}_\mu$ NC[1]. For validation and test, a sample of $\sim$ 60.000 events has been used: in this case relative abundances are strongly limited by the structure of the .root files. In particular, a proportion of 30% of track-like events and 70% of shower-like events has been used (internal proportions kept as before).

4 Structure of the codes

The general structure of the scripts can be summarized as follows:

1. **File preparation.**
   (a) **Orcasong** is a project for preprocessing raw KM3NeT/ARCA event data for the use with deep neural networks[2]. All the related codes can be found at GNN/SCRIPTS/ORCASONG folder.
   (b) Root files are converted into a .h5 format via H5_EXTRACT_ALL.SH, which takes as an input a LIST.TXT where files names are written. SUBMIT_H5EXTRACT.SH submits the job to a cluster.
   (c) .h5 files are converted into a format specificly desingned from deep learning algorithms, called root_dl.h5. This is done via some Orcasong extractors specified in EXTRACT_ALL.PY and again submitted to a cluster with .sh scripts. Until now, the steps are equally applied to test, validation and training.
   (d) For the training the concatenate step is needed: it creates a single file with all the events. SUBMIT_CONCATENATE.SH needs a list in format .txt with the names of deep learning .h5 files, alongside the name of the output file.
   (e) Finally, for the training it is necessary submit event packages in a random order. This is done with SHUFFLE.SH which is submitted to a cluster with SUBMIT_SHUFFLE.SH.
2. **Training and validation.**
   (a) Training part is developed in the context of **Orcanet**, a deep learning framework which simplifies the training process of neural networks for astroparticle physics. It incorporates automated logging, plotting and validating during the training, as well as saving and continuing the training process[3].

[1] Previous studies were suggesting to limit the presence of hadronic showers in the training sample because are quite disturbing events.
[2] https://ml.pages.km3net.de/OrcaSong/index.html
[3] https://ml.pages.km3net.de/OrcaNet/index.html

(b) Folder GNN/SCRIPTS/TRAINING contains two .sh scripts where a singularity image (orcanet_v1.0.4.sif) is specified in order to run Orcanet.

(c) GNN/SCRIPTS/TRAINING/TS contains the .toml files:

i. **list.toml**, where the data set, used for training and validation, is specified.

ii. **config.toml** configures the training. The learning rate can be submitted in tuple format of the type [float1,float2], where float1 indicates the learning rate for epoca1 and file1, while float2 indicates the decreasing percentage of float1 in subsequent files. In this analysis learning_rate = [0.003,0.001].

iii. **model.toml** defines the model architecture.

(d) submit_train.sh indicates epochs and other parameters needed for saving the outputs. The output can be found at GNN/ARCA21/TS_ONLYNU_CORRECTED/WITH16NEIGHB, where the SUMMARY.TXT file contains, for each epoch, loss, accuracy and learning rate. The models containing the parameters of the networks are stored in ../SAVED_MODELS and, finally, ../PLOTS folder contains the output plots of loss and accuracy.

3. **Test (or inference).**

(a) GNN/SCRIPTS/INFERENCE contains the .sh and .py scripts for the test, which is done on a .h5 file produced after the concatenation of root_dl.h5 files. The output is a .h5 (namely ARCA21_TS.h5) file containing the scores of the events. The code automatically selects the epoch with the lowest loss at validation and takes the associated network parameters among the saved models.

4. **Final plots.**

(a) Final plots are produced with results.py script stored in GNN/ANALYSIS folder.

5 Loss and accuracy

The loss function adopted for the network is a categorical cross entropy. The training of the sample was done on a GPU and lasted 7 days, after which the algorithm was stopped for computing resource reasons. It reached 13 epochs. Each epoch required almost 12 hours for the training and 1 hour and half for the validation: the validation was done with the trained network at the end of each epoch, considering the parameters of the model stored at that the end of the training. Accuracy, instead, is defined as the ratio between events correctly classified (considering a reference threshold at 0.5 for the score) and the total number of events. Plots for loss and accuracy are shown in Figure 2 and Figure 3 respectively. Even if the code stopped at epoch 13, we observe a saturation in both the evolution of the loss and the accuracy during the training. The trend of the validation is quite unexpected and has a regular pattern: every 3 epochs there is bump in the loss (so a drop in the accuracy), which is then quickly recovered at the following epoch. This behaviour is still under study: it has already been cross-checked that no dropout was active in the algorithm. Also the option that it can be due to a disproportion of relative abundances between training and validation

samples has been tested without success. Anyway, this has brought to the epoch 13, where observe the best results so far: a validation loss of 0.22 and a validation accuracy of 0.92. After one week of training, this is has been chosen as best model of the network and used for the test.

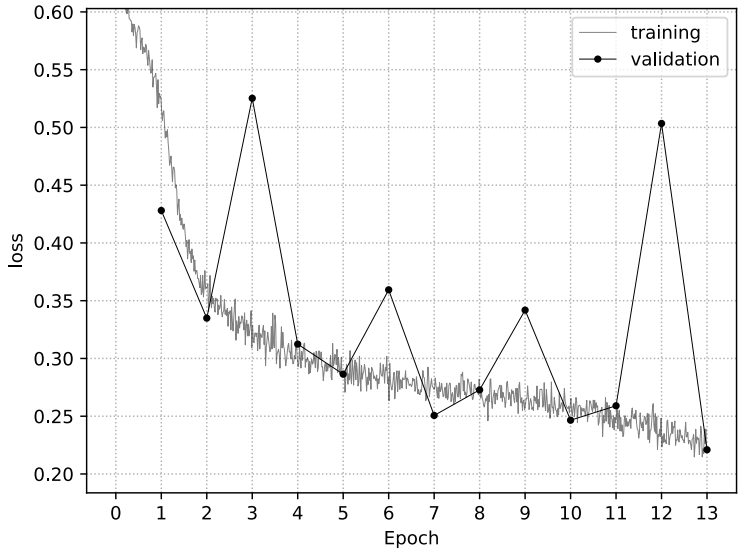


Figure 2: Training history of the neural network used for tracks and showers classification. Continuous line is referred to the loss during the training, where batches of events are submitted to the network. Points are the validation results at the end of the epoch.

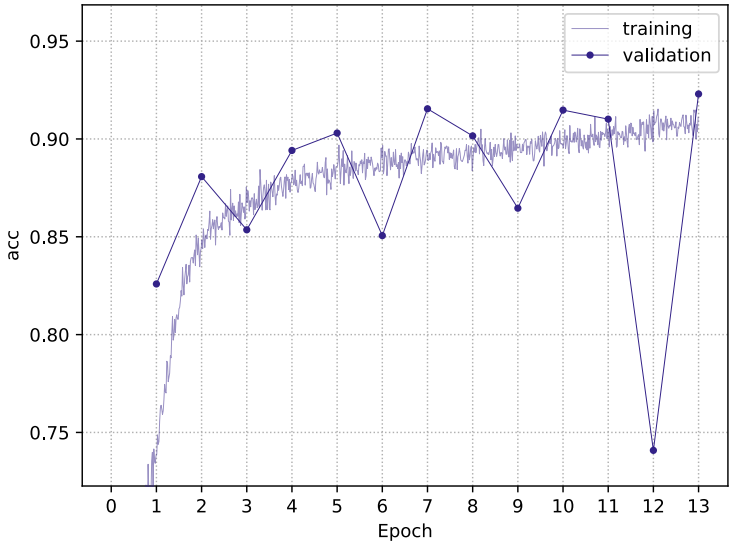


Figure 3: Accuracy history of the neural network used for tracks and showers classification. Continuous line is referred to the accuracy during the training, where batches of events are submitted to the network. Points are the validation results at the end of the epoch.

6 Results

The result of the test is the shower score distribution, visible in Figure 4. The basic concept, in order to choose the classification threshold, is to maximize the rate of events belonging to the positive class (showers), correctly classified, and minimize the rate of mis-classified events. For every possible value of the shower score, the integral (fraction of events) on the right side of the light-blue distribution is calculated, giving back the so-called True Positive Rate (TPR). In the same way, the integral of the orange curve in the same region gives back the False Positive Rate (FPR), so the events belonging to the shower class even if are not shower-like events. This calculation, repeated for every score values, gives a plot of TPR vs FPR (Figure 5). The best value of the cut is found to be the one that maximizes the so-called G-mean function:

$$\textbf{G-mean} = \sqrt{\text{TPR}\ (1 - \text{FPR})}$$

This is the ROC curve (Receiver Operating Characteristic curve) method and should provide a balance between quality of the classification and number of events correctly classified. This cut is the separation line between the two classes of events, found at 0.697. Figure 6 shows the TPR and FPR values, in percentage, as a function of the shower score cut.

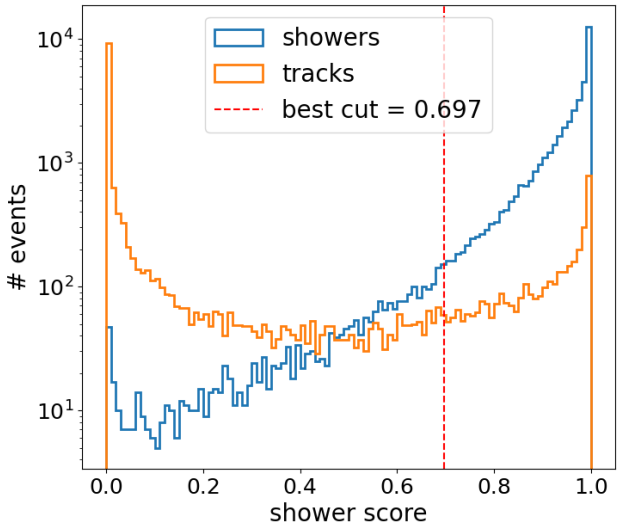


Figure 4: Shower score distribution for tracks and showers classification. A best cut of 0.697 is found to be the optimal separation between the two classes of events.

Figure 7 shows the same results of Figure 4, reporting on the x-axis the track score, with respect to which the cut has been optimized. This leads to a best cut of 0.303, complementary to what has been obtained for the shower score. Previously in the Collaboration, deep-learning classifications with ARCA21 have been done considering also atmospheric muons simulations in the training sample (a factor 19% of the total track-training, relying on an older MC version (8.0)). A model trained also with atmospheric muons has been tested

on a sample containing only $\overset{(-)}{\nu}_{\mu}$ CC and shower-like events (Figure 8), leading to a best cut of 0.475. This is directly comparable with this analysis (Figure 7), where only $\overset{(-)}{\nu}_{\mu}$ CC interactions have been considered for the track-training sample. It is possible to conclude that a training performed only with neutrino simulations is more effective in distinguishing shower-like events. Thanks to the lowering of the best-cut level, showers can therefore be more safely separated from tracks than before.

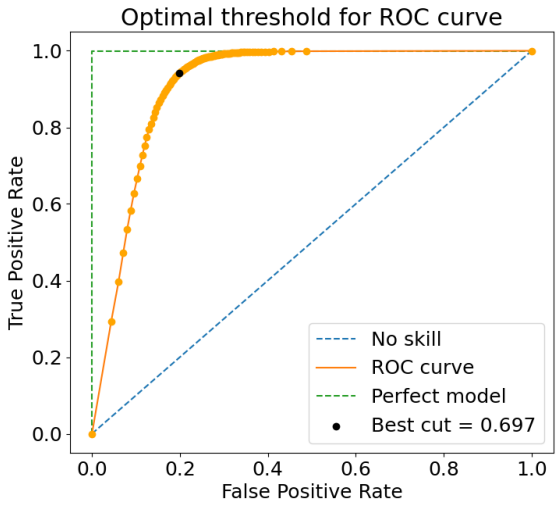


Figure 5: The ROC curve method. For each score value, a yellow dot is plotted and corresponds to a given fraction of TPR and FPR. Black dot represents the score value that maximizes the G-mean. Green line is referred to an ideal model with no FPR and 100% of TPR. The "no skill" blue curve is the line we should have obtained if the score distributions were flat.

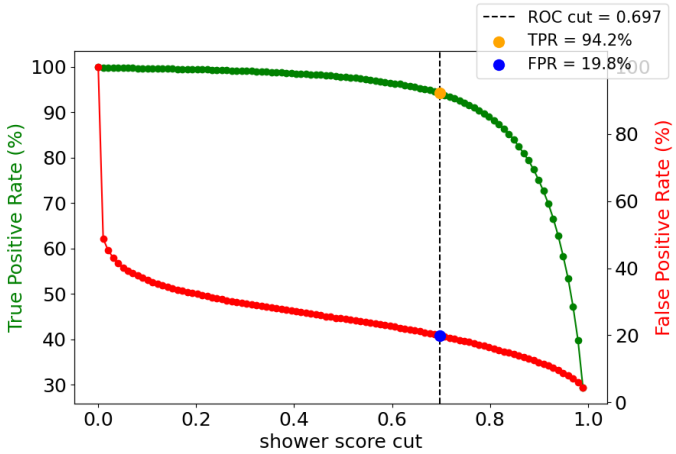


Figure 6: TPR and FPR, in percentage, as a function of the shower score cut. Their values in correspondence of the best cut are also reported.

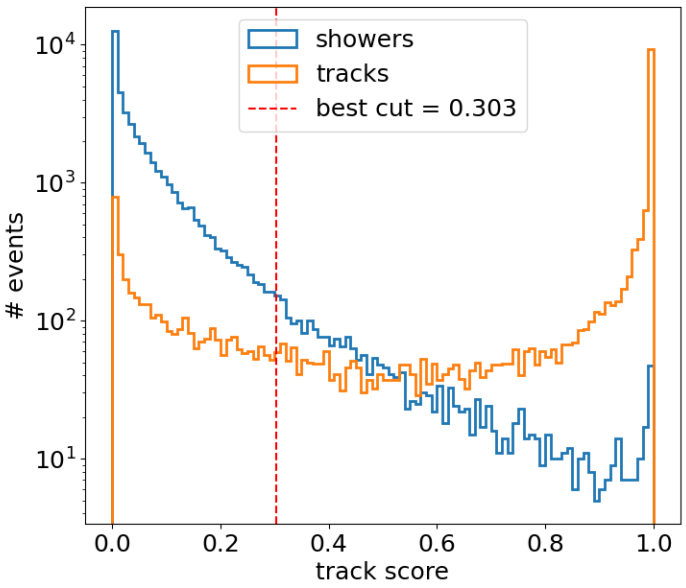


Figure 7: Track score distribution in this analysis. A best cut of 0.303 is found to be the optimal separation between the two classes of events.

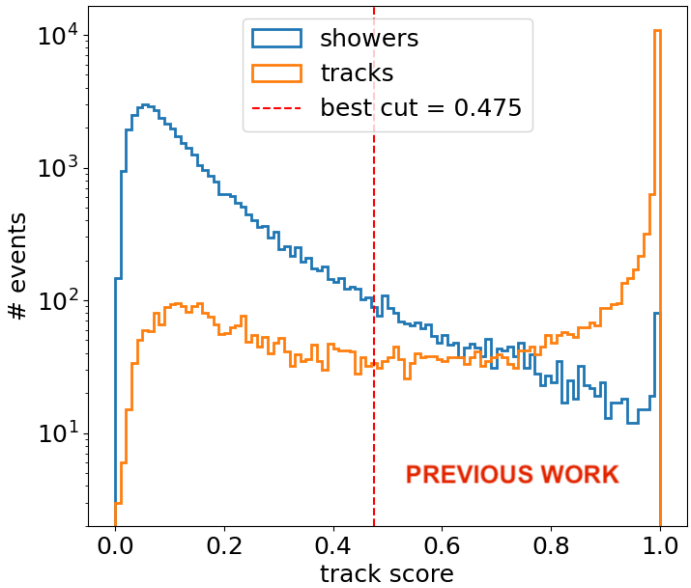


Figure 8: Track score distribution for tracks and showers classification with a training sample containing also atmospheric muons, evaluated on the test sample of this analysis. A best cut of 0.475 is found to be the optimal separation between the two classes of events.

The ultimate comparison is performed between the best model of this analysis, valuated on a test sample containing atmospheric muons, $\overset{(-)}{\nu}_{\mu}$ CC and shower-like events (Figure 9), and the same best model of Figure 8, valuated on the test sample used in Figure 9 (see Figure 10). A detailed view of the relative abundances contained in this test sample is shown in Figure 11. Again, a lowering of the best cut threshold is observed when a model trained

with only neutrino-interaction simulations is used.

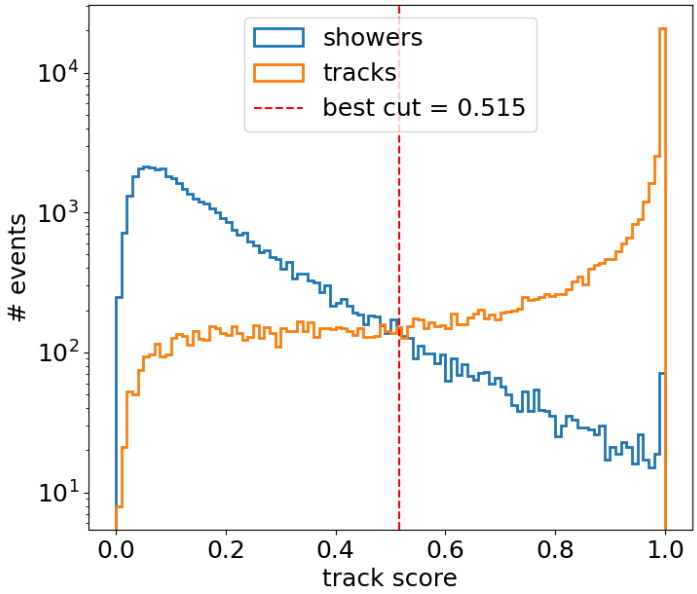


Figure 9: Track score distribution for tracks and showers classification considering the training of this analysis, whose best model is valuated on a test sample containing atmospheric muons, $\overset{(-)}{\nu}_{\mu}$ CC and shower-like events (Figure 11).

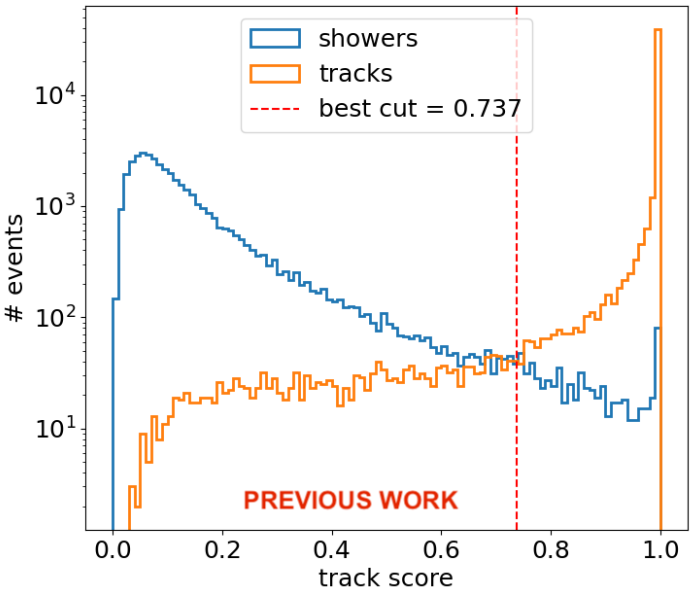


Figure 10: Track score distribution for tracks and showers classification considering a training sample containing also atmospheric muons, whose best model is valuated on a test sample containing atmospheric muons, $\overset{(-)}{\nu}_{\mu}$ CC and shower-like events (Figure 11).

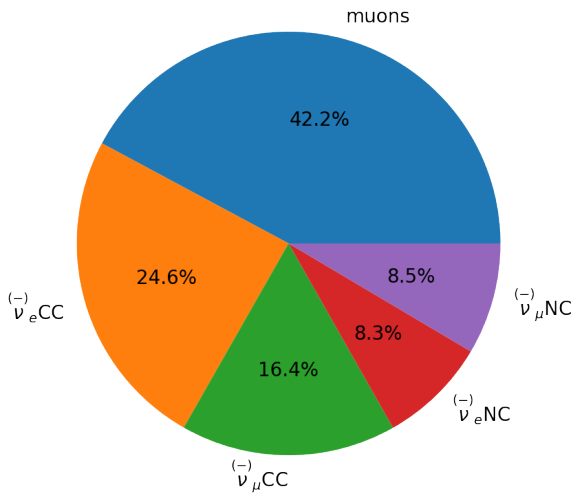


Figure 11: Relative abundances of the test sample used for Figures 9 and 10.

References

[1] Huilin Qu and Loukas Gouskos. "ParticleNet: Jet Tagging via Particle Clouds". In: *Phys. Rev. D* 101.5 (2020), p. 056019. DOI: 10.1103/PhysRevD.101.056019. arXiv: 1902.08570 [hep-ph].

Acknowledgements

Thank you to Dr Alessandro Veutro of the KM3NeT-Rome group for all the precious suggestions.