Machine Learning: Science and Technology

DeepRICH: learning deeply Cherenkov detectors

IOP Publishing Mach. Learn.: Sci. Technol. 1 (2020) 015010 https://doi.org/10.1088/2632-2153/ab845a

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
10 January 2020

REVISED
11 March 2020

ACCEPTED FOR PUBLICATION
27 March 2020

PUBLISHED
27 April 2020

PAPER

DeepRICH: learning deeply Cherenkov detectors

Cristiano Fanelli[1,3] **and Jary Pomponi**[2]

[1] Laboratory for Nuclear Science, Massachusetts Institute of Technology, Cambridge, MA 02139 United States of America
[2] Department of Information Engineering ,Electronics and Telecommunications Sapienza University of Rome, Italy
[3] Jefferson Lab, EIC Center, Newport News, VA 23606, United States of America

E-mail: cfanelli@mit.edu

Keywords: Cherenkov detectors, deep learning, particle identification, near real time algorithm, experimental particle physics

Abstract

Imaging Cherenkov detectors are largely used for particle identification (PID) in nuclear and particle physics experiments, where developing fast reconstruction algorithms is becoming of paramount importance to allow for near real time calibration and data quality control, as well as to speed up offline analysis of large amount of data.

In this paper we present DeepRICH, a novel deep learning algorithm for fast reconstruction which can be applied to different imaging Cherenkov detectors. The core of our architecture is a generative model which leverages on a custom Variational Auto-encoder (VAE) combined to Maximum Mean Discrepancy (MMD), with a Convolutional Neural Network (CNN) extracting features from the space of the latent variables for classification.

A thorough comparison with the simulation/reconstruction package FastDIRC is discussed in the text. DeepRICH has the advantage to bypass low-level details needed to build a likelihood, allowing for a sensitive improvement in computation time at potentially the same reconstruction performance of other established reconstruction algorithms.

In the conclusions, we address the implications and potentialities of this work, discussing possible future extensions and generalization.

1. Introduction

Imaging Cherenkov detectors [1] measure the velocity of charged particles and, if combined with independent measurements of their momentum, are largely used for PID in modern particle physics experiments. The photon pattern recognition is typically likelihood-based and requires computationally expensive simulations, hence different strategies (among which pre-computed look-up tables) have been developed to find a trade-off between time and reconstruction performance. A particular class of Cherenkov detectors is based on the detection of internally reflected Cherenkov (DIRC) light (see, *e.g.* [2]): light is contained by total internal reflection inside a solid radiator preserving its angular information until it reaches spatially segmented photon sensors, where typically rather complex hit patterns are observed.

Machine learning (ML) algorithms are already the state-of-the-art in event and particle identification in high energy physics [3] but solutions based on ML for this kind of detectors just started being explored [4].

The first DIRC detector was developed by the BaBar experiment at SLAC [5], and inspired other experiments (see, *e.g.* [6–8]) to utilize similar detectors, also in view of future experiments like the Electron Ion Collider [9]. In the following we will consider as an example the case of the GlueX experiment [2, 10] at the Jefferson Laboratory, where the DIRC has been recently installed utilizing components of the decommissioned BaBar DIRC to enhance the PID capabilities of the experiment. Our choice is motivated by FastDIRC [11], an open source simulation and reconstruction package for DIRC detectors implementing the GlueX DIRC geometry.

This geometry consists of four bar boxes and two photon cameras, where each bar box contains 12 fused silica radiators ($1.725 \times 3.5 \times 490$ cm^3). Both photon cameras are attached to two bar boxes and are equipped with an array of Multianode Photomultiplier Tubes (MaPMTs) allowing a three-dimensional

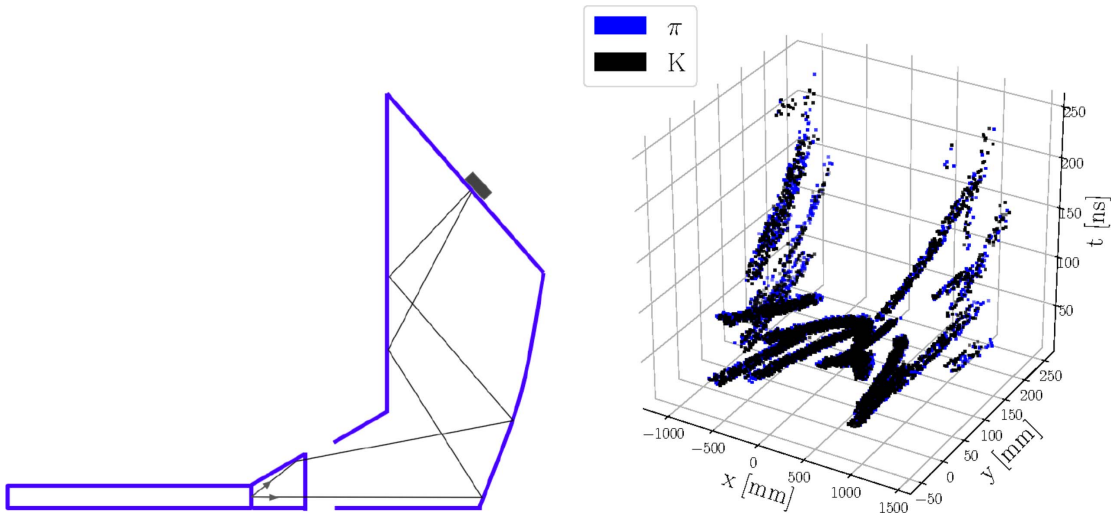

Figure 1. (left) Drawing (not to scale) of the GlueX DIRC box containing fused-silica bars and wedges. An optical box filled with distilled water is used to transport the Cherenkov photons from the wedges to the PMT arrays. (right) Example of hit pattern detected in the PMT plane (spatial coordinates are dubbed x,y, while the time is indicated as t) simulated with FastDIRC. The two colors correspond to the hit pattern of a kaon and of a pion as reported in the legend, under the same kinematic conditions (i.e. particle momentum, incident angle on the bar, azimuthal angle with respect to the bar, location on the bar and which bar has been hit).

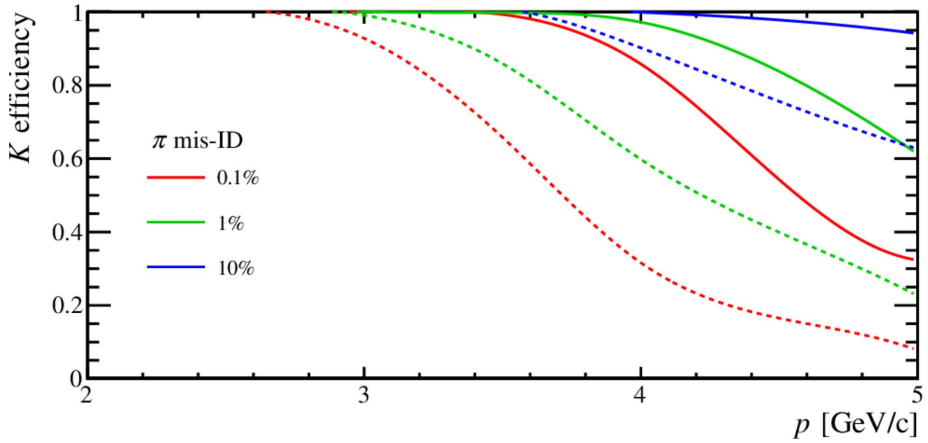

Figure 2. Kaon efficiency vs momentum for π mis-identification probabilities of 0.1, 1, and 10% (i.e. probability for a charged π track to be incorrectly identified as a charged K). The dashed curves show the conservative performance, while the solid curves show the improved performance achieved in simulation for the current GlueX DIRC design. Image taken from [2], where the reader can find more details.

(x, y, t) readout with a time resolution of approximately 200 ps. Patterns take up significant fractions of the PMT in x,y and are read out over 50-100 ns due to propagation time in the bars. The reader can find in figure 1 (left) a schematic of the detector with one of the two photon cameras and in figure 1 (right) an example of hit pattern generated with FastDIRC expected in the PMT plane (x,y) as a function of the propagation time. In particular, the GlueX experiment is designed to search for gluonic excitations in the meson spectrum produced through photoproduction reactions at a tagged photon beam facility. For this physics program, the DIRC is expected to provide a good separation power between pions and kaons of at least 3σ up to 4 GeV/c in momentum (a plot of the kaon efficiency as a function of the kaon momentum for different pion mis-identification probabilities is shown in figure 2), which allows systematic studies of kaon final states that are essential for inferring the quark flavor content of both hybrid and conventional mesons [12, 13]. For all these reasons, developing an efficient and fast reconstruction algorithm is of crucial importance. Notice that in the case of ring imaging Cherenkov (RICH) detectors, the time variable is typically not used in the reconstruction methods. This feature could be part of future reconstruction algorithms if better time resolutions are achieved. Instead in the DIRC case, the larger propagation times contribute to distinguish the type of particle producing Cherenkov light. Depending on the type of detector, DeepRICH reconstruction can be based on spatial features only or on combined space and time components.

The outline of this paper is as follows: existing reconstruction methods are discussed in section 2; the DeepRICH architecture is presented in section 3; application to the DIRC case, discussion of the results and comparison with FastDIRC are described in section 4; summary and conclusions are reported in section 5.

2. Established methods and novel approaches

Cherenkov detectors are relatively slow to simulate with full simulations like Geant [14]—*e.g.*, for the DIRC case, each Cherenkov photon reflects on average $\mathcal{O}(10^2)$ times within a bar and this makes the simulation CPU intensive—thus new approaches are being developed to get a faster reconstruction of the detected light [4, 11, 15, 16]. In this section we briefly describe the state of the art of established computational methods and provide an overview of novel paradigms based on machine learning.

2.1. The geometrical reconstruction method

The geometrical reconstruction method is based on the BaBar DIRC algorithm [17]. This approach involves generating in advance a large number of photons at different angles exiting each bar, and then tracking them to the PMT plane. In this way the look-up table is created, where each pixel on the photo-detection plane is associated to a set of photon directions at the exit from the bar potentially leading to a photon detected in that pixel. The Cherenkov angle θ_C of each photon is then reconstructed combining the particle direction provided by the tracking system with the photon direction taken from a look-up table. The look-up table is stored as a ROOT tree with the size of about 300 MB [10]. The resulting cumulative distribution of the reconstructed Cherenkov angles is typically characterized by peaks at the expected values of θ_C for pions and kaons and a combinatorial background beneath them. The width of the Cherenkov angle reflects the single photon Cherenkov angle resolution characteristic of the detector performance.

2.2. Time-based image reconstruction

Another approach is the so called time-based imaging reconstruction which is derived from a method used by the Belle II TOP [18]. For every particle hypothesis, the expected arrival time of Cherenkov photons is calculated analytically based on the charged particle direction and hit location and is compared to the measured time, yielding to likelihoods. This method is rather compute-intensive, as one in principle should simulate all the configurations of the charged particles as a function of the mass, energy, direction and location in the DIRC bars.

2.3. FastDIRC

The main characteristic of the FastDIRC algorithm [11] is to analytically trace the photons through the optical system. This approach is about $\mathcal{O}(10^4)$ times faster than the full Geant simulation. The reconstruction is based on a kernel density estimation (KDE) [19] of the probability distribution function (PDF) for each assumed particle type. The expected distributions on the detection plane for each charged particle hypothesis are compared to the actually observed hit patterns to build likelihoods. FastDIRC allows for parameterization, a feature that makes it suitable for detector design optimization and for offline calibration of real data. It has been shown [11] that the resolution of the reconstructed Cherenkov angle is about 30% better than the geometric reconstruction method. However the FastDIRC method is about $\mathcal{O}(10^2 - 10^3)$ times slower than the look-up table based reconstruction.

In this paper, FastDIRC is used as a source of reliable simulated events that are injected as input of the DeepRICH architecture.

2.4. Generative adversarial network

A first attempt to apply deep learning to simulate Cherenkov detector response appeared recently in [4], where it has been proposed to use a generative adversarial neural network (GAN) [20] to bypass low-level details at the photon generation stage. This work is based on events simulated with FastDIRC assuming the design of the GlueX DIRC. The GAN architecture is trained to reproduce high-level features (the likelihood results from FastDIRC) based on input observables of the incident charged particles, allowing for an improvement in simulation speed. The authors of [4] claim a good precision and very fast performance (the batch generation on GPU produces up to 1 million track predictions per second) from their studies.

Recently in another paper [21] generative models have been used for fast simulation of RICH detectors at LHCb.

In the following section we are going to present a new deep architecture called DeepRICH, providing a thorough description of the code, data preparation, training/testing phases and performance.

3. The deepRICH network

Differently from the GAN based method, which directly maps the injected input to the reconstructed output, our generative model explicitly reconstructs the injected hit patterns expected for each kinematics, and internally creates latent variables that allow to classify the particles.

3.1. Architecture

DeepRICH is based on a custom Variational Auto-encoder [22]. VAEs are generative models that try to simulate how the data are generated. In order to characterize the causal relations underlying the observed data, VAEs provide a posterior function approximated by an autoencoder architecture, which is made by an encoder and a decoder, the latter being symmetric to the first in terms of layer structure.

In what follows we describe each detected hit by a three-dimensional vector, (x,y,t), corresponding to the spatial and temporal components. We use the notation $\mathbf{x} \in \mathbb{R}^{m\times 3}$ to indicate m hits associated to an individual charged particle. The kinematic parameters of each particle are represented by the vector **h**, and they embody information on the particle momentum, angle and location where the particle crossed each bar (more details on this can be found in section 3.2, where we discuss about the preparation of data).

Our novel architecture consists of three main parts:

- An **Encoder**, which takes as input the concatenation between (i) m hits produced by a particle, $\mathbf{x} \in \mathbb{R}^{m\times 3}$ and (ii) the associated vector **h** of kinematic parameters, to produce a d-dimensional vector of latent variables for each input hit, i.e. $\mathbf{l} \in \mathbb{R}^{m\times d}$.
  These vectors contain all the information that the network is capable of extracting from the hits **x**.
- A **Decoder**, which takes as input the vectors of latent variables **l** concatenated with **h** and provides as output a set of hits $\tilde{\mathbf{x}} \in \mathbb{R}^{m\times 3}$, corresponding to the reconstruction of the input **x**.
- A **Particle Classifier**, which basically consists in convolutional and linear layers; the network takes as input the vectors of latent variables **l** to classify the particle.
  The challenging aspect here is to use the information extracted from the Encoder to do PID, that is to understand if the particle that has generated the hits $\mathbf{x} \in \mathbb{R}^{m\times 3}$ is a pion (π) or a kaon (K)[1].

A flowchart of the DeepRICH network is represented in figure 3.

The model is trained by minimizing the total loss function which is:

$$\begin{aligned}&\mathcal{L}(\mathbf{x},\tilde{\mathbf{x}},\mathbf{y},\tilde{\mathbf{y}},\mathbf{l}) = \\ &\lambda_r\mathcal{L}_r(\mathbf{x},\tilde{\mathbf{x}}) + \lambda_c\mathcal{L}_c(\mathbf{y},\tilde{\mathbf{y}}) + \lambda_v\mathcal{L}_v(\mathbf{l}),\end{aligned} \tag{1}$$

where the λ multipliers are used to weigh the contribution of the corresponding loss terms, described in the following:

(i) The term $\mathcal{L}_r$ is the average reconstruction loss between the real particle **x** and the output of the Decoder $\tilde{\mathbf{x}}$, calculated using the L1 smooth loss (also called Huber error. See, *e.g.*, [24])[2]:

$$\mathcal{L}_r(x,\tilde{x}) = \frac{1}{3}\sum_i^3 z_i \tag{2}$$

where z_i is given by

$$z_i = \begin{cases} 0.5(x_i-\tilde{x}_i)^2, & \text{if } |x_i-\tilde{x}_i| < 1 \\ |x_i-\tilde{x}_i| - 0.5 & \text{otherwise,}\end{cases}$$

and the index i indicates the spatial or time components of each hit. Such a loss is less sensitive to outliers than the Mean Squared Error (MSE). In fact in the case of an unbounded output, MSE requires careful tuning of the learning rate and the loss in order to prevent exploding gradients.

(ii) The term $\mathcal{L}_c$ is the classification accuracy, calculated using the Cross Entropy between the target y, i.e. the ground truth particle's type (0 for kaons and 1 for pions), and the output of the classification layer $\tilde{y}$.

$$\mathcal{L}_c = -(\mathbf{y}\log(\tilde{\mathbf{y}}_0) + (1-\mathbf{y})\log(\tilde{\mathbf{y}}_1)) \tag{3}$$

[1] We are focused here on distinguishing πs from Ks as this is the main scope of the GlueX DIRC. DeepRICH can be generalized to more than two categories of particles.

[2] Notice we are using a simplified notation: another sum running over all the hits of the particle is present in equation (2).

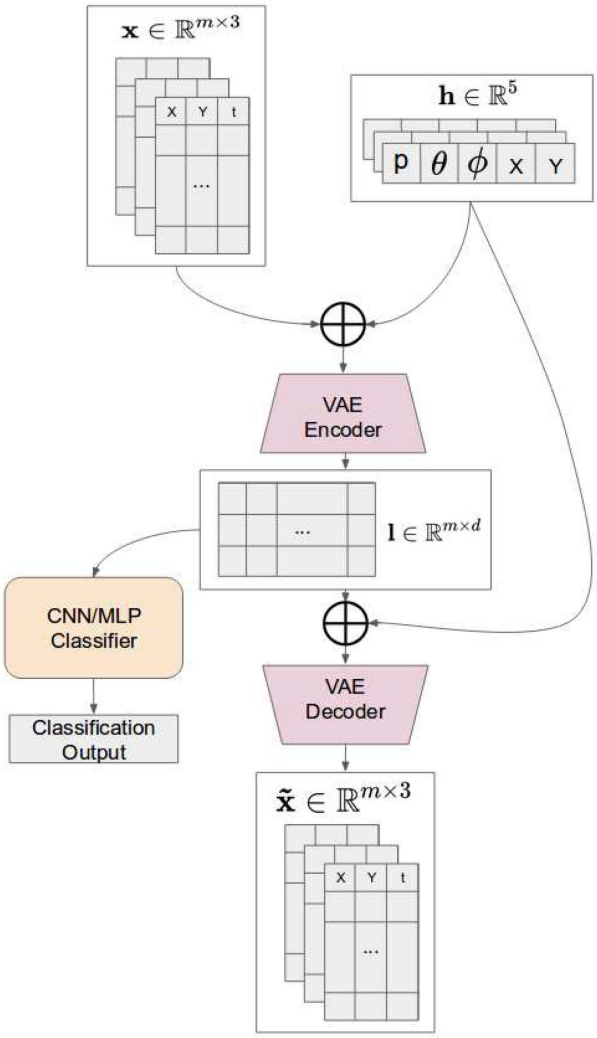

Figure 3. A flowchart of DeepRICH: the inputs are concatenated—n.b., the $\oplus$ represents the concatenation between vectors—and fed into the encoder, which generates a set of vectors of latent variables, which are then used for both the classification of the particle and for the reconstruction of the hits. Image taken from [23]

where the components $\tilde{\mathbf{y}}_0$ and $\tilde{\mathbf{y}}_1$ refer to the logits, associated to pions and kaons, scaled using the *softmax*$(\cdot)$ function. After this scale we have $\tilde{\mathbf{y}}_0 + \tilde{\mathbf{y}}_1 = 1$.

(iii) The loss $\mathcal{L}_v$ is a term calculated using the Maximum Mean Discrepancy (MMD) [25], as explained in the following; notice that the idea of combining VAE and MMD was used for the first time in [26], where the authors proved that infoVAE (VAE using MMD) is fast to train, stable and leads to a better learning of the features if compared to the traditional evidence lower bound (ELBO) [27] criterion used in VAEs. The basic idea of MMD is that two distributions are identical if and only if their moments are the same. Assuming to have two distributions $\boldsymbol{p}(z)$ and $\boldsymbol{q}(z)$, one can measure the divergence between these distributions:

$$\begin{aligned}\mathcal{L}_v = MMD(\boldsymbol{p}(z), \boldsymbol{q}(z)) = \mathbb{E}_{\boldsymbol{p}(z),\boldsymbol{p}(z')}[\kappa(z,z')] \\ + \mathbb{E}_{\boldsymbol{q}(z),\boldsymbol{q}(z')}[\kappa(z,z')] - 2\mathbb{E}_{\boldsymbol{p}(z),\boldsymbol{q}(z')}[\kappa(z,z')],\end{aligned} \tag{4}$$

where $\kappa(\cdot,\cdot)$ can be any positive definite kernel, which can be seen as a function that measures the distance between two samples. To this end, we use a Gaussian kernel [28]. In our case the distribution $\boldsymbol{p}(z)$ is related to the vector of latent variables, and $\boldsymbol{q}(z)$ is a normal distribution $\mathcal{N}(0,\sigma)$; the best value of σ is determined using the Bayesian optimization described in section 3.4. A naive intuition of MMD is that the latent vectors should follow the same distribution of $\boldsymbol{q}(z)$. The architecture described in this section is also summarized in form of a pseudo-code in the algorithm 1.

Table 1. A detailed description of the DeepRICH architecture used in the experiments. Each sub-architecture—specified by the type of neural network—is described layer by layer in terms of the number of neurons and the size of the kernel, the used activation function and the regularization.

Architecture name	Type	Neurons/kernel size	Activation function	Regularization
Encoder/Decoder	Linear	512/256	ReLU	Drop 0.1
		256/512	ReLU	Drop 0.1
		Latent Dim/3		
CNN	Conv	64 (3x3, Stride 1)	ReLU	Batch Norm
		64 (3x3, Stride 1)	ReLU	Batch Norm
		128 (3x3, Stride 1)		Batch Norm + Drop 0.5
Classifier	Linear	100	ReLU	
		50	ReLU	
		25	ReLU	
		2		

Algorithm 1 Pseudo-code for particle identification with DeepRICH.

proceduretraining
for$i = 1 \ldots$E (training epochs)
foreach batch $b = (\mathbf{x}, \mathbf{y}, \mathbf{h}) \in$ B
$\mathbf{l} \leftarrow$ *Encoder*$(\mathbf{x}, \mathbf{h};\ \theta)$
$\tilde{\mathbf{y}} \leftarrow$ *Classifier*$(\mathbf{l};\ \theta)$
$\tilde{\mathbf{x}} \leftarrow$ *Decoder*$(\mathbf{l}, \mathbf{h};\ \theta)$
update θ by minimizing total loss $\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}, \mathbf{y}, \tilde{\mathbf{y}}, \mathbf{l})$
end for
end for
end procedure

- $\mathbf{x} \in \mathbb{R}^{m \times 3}$ is a set of hit produced by a charged particle; $\mathbf{y}$ is the ground truth of the particle (i.e. a π or a K); $\mathbf{h}$ is the vector containing the kinematic parameters associated to the particle; θ are the weights (parameters) of the networks; $\mathbf{l}$ is the vector of latent variables associated to $\mathbf{x}$ and produced by the Encoder.
- For each hit in the particle, the Encoder produces a vector of the latent variables $\mathbf{l}$, by taking as input the encoded kinematic parameters concatenated with the hit itself.
- The vectors of latent variables associated to the hits of a particle are used to classify the particle itself.
- The Decoder reconstructs the input hits using the latent variables and the kinematic parameters.

In addition we use a dropout layer after each layer in the Encoder/Decoder, with drop probability equal to 10%; we also apply a dropout on the latent variables before feeding them into the CNN, with a probability equal to 50%. We fix the number of layers in the decoder/encoder to 2, while the number of hidden units is set to [512, 256]. The CNN has 3 layers with, respectively, [64, 64, 128] kernels with stride 1 and size 3, whereas the classifier has 4 layers with [100, 50, 25, 2] neurons, where the dimension of the last layer correspond to the number of classes (π and K). The activation function used after each layer is the Rectified Linear Unit (ReLU). The reader can find more technical details summarized in table 1.

3.2. Data Preparation

The data generation is based on FastDIRC [11]. FastDIRC allows to generate the hit pattern observed in the PMT detection plane for a given kinematics of the charged particle traversing the radiator. The kinematics is characterized by different parameters, namely the momentum of the particle p [GeV/c], the polar angle θ relative to the normal to the surface of the bars, the azimuthal angle ϕ, the location X, Y on the surface of the bar, the information (as an integer index) on which fused silica bar has been hit [3]. FastDIRC uses kernel density estimation to produce an estimate of the probability distribution function on the PMT plane. It generates about 10^5 provisional points for each kinematics, which are used to detect an actual charged particle passing through the bars and generating a sparse hit pattern of about 20-50 'real' hits. FastDIRC therefore generates both the sparse hit patterns associated to one particle as well as the whole probability density function (PDF) associated to a particular kinematics which is used to identify that particle. The training set for DeepRICH has been generated with FastDIRC combining more than one kinematics for a single bar. A particular region of the phase-space can be divided into a fine grid of points. For example, the

[3] We use capital letters to distinguish the location on the bar (X,Y) from the hit spatial coordinates (x,y) in the detection plane.

largest dataset we generate corresponds to an hypercube of $\sim 2{\cdot}10^4$ kinematic points covering the kinematic subspace $\Delta p \times \Delta\theta \times \Delta\phi \times \Delta X \times \Delta Y = [4, 5]$ [GeV/c] $\times$ [2, 5] [deg] $\times$ [20, 90] [deg] $\times$ [-17.5, 17.5] [mm] $\times$ [50, 1000] [mm], where θ, ϕ, X and Y have been divided into equally distant points within those intervals.

For each point of the grid we generate one PDF with FastDIRC and then sample randomly the observed 'real' hits. This is done by taking into account the expected yield of the photons: we implemented a yield generation inspired by the FastDIRC simulation of the observed hits which takes into account the photon yield reduction due to several effects, e.g., if the total internal reflection condition is not met or a photon misses a mirror. We also check that keeping the yield constant (fixing it to 40 photons) does not change the performance significantly.

Consistently with the expectations, a more dense grid of points combined with a larger number of sampled particles at each kinematic point generally improves the PID performance of DeepRICH (this can be quantified as the Area Under Curve described in section 4.1). Taking into account that the intrinsic limit on the achieved performance depends on the kinematic conditions (*e.g.*, the larger the momentum the lower is the π/K distinguishing power), a tradeoff on the above numbers (i.e. how dense the grid and how many particles should be chosen for training) can be found based on the sought classification accuracy and the available computing resources.

3.3. Model training and testing

At each kinematic point (p, θ, ϕ, X, Y) we use FastDIRC to produce a large number of expected hits for both πs and Ks. Then we sample N particles of a given type (π or K) where by construction each particle consists of a random set of m hits. In this way we avoid that the network learns how to classify particles based on some patterns internal to the FastDIRC generation algorithm. At the same time with this choice we can virtually build an unlimited dataset of particles from the PDFs of FastDIRC.

The generated samples have been then divided into two subsets: training and test: (i) The training set contains particles at certain kinematics which are used during the training phase—ensuring that all the vertices of the hypercube are included—while (ii) the test subset will be used only for testing the network performance after the training procedure to see if it can achieve good results on unknown kinematics. Furthermore the particles from the training set are divided into 'training particles' and 'development particles' (the split is 80%/20%); the training particles are used to update the parameters of the network by minimizing the total loss (see equation (1)), while the development particles are used to calculate an accuracy score, to evaluate the goodness of the classification while training and check if the network is learning properly how to classify hits from known kinematics. Early stopping is used to interrupt the training procedure if the development score does not improve after a certain number of epochs. The classification score on the development particles is also used to tune the hyperparameters of the network with a Bayesian optimization (the procedure is explained in detail in section 3.4).

We then optimize the parameters of the network with Adam [29] using the tuned learning rate. The dataset has been standardized—for each feature we choose 0 mean and standard deviation (Std) equal to 1—and this is done separately for both the hits and the kinematics parameters, in order to avoid the overshadowing of features with smaller values and further improve the training procedure; notice that the development and test hits have been standardized using the mean and the Std calculated on the training hits, to avoid a potential injection of bias that could improve the classification performance.

We train the network in different experiments, each consisting of at most 50 epochs, and evaluate the performance on the development subset during the training phase. The development accuracy is calculated by applying the $softmax(\cdot)$ on the classification layer. When the training is over, the model is evaluated on the test particles extracted from unknown kinematics.

3.4. Network optimization

Bayesian Optimizers (BOs) [30, 31] are among the most efficient tools for optimizing the hyperparameters of a deep architecture [32]. In fact BOs search for the global optimum x^* over a bounded domain χ of a black-box functions $f(x)$. In particular, f can be noisy, non-differentiable and expensive to evaluate.

Typically Gaussian processes [33] are used to build a surrogate model of f, but other regression methods such as decision trees can also be used. Once the probabilistic model is determined, a cheap utility function (also called acquisition function) is considered to guide the process of sampling the next point to evaluate. The DeepRICH network consists of N hyperparameters listed in table 2. In particular, the multipliers of the loss functions defined in equations (2), (3), (4), the dimension of the latent variables, the MMD variance and the learning rate play an important role in the performance of the network. These hyperparameters are tuned with a BO provided by the sklearn [34] package. As previously discussed, other hyperparameters, *e.g.*, the number of layers in the architecture, are not tuned and their values are reported in table 1. We choose as

Table 2. List of hyperparameters tuned by the BO. The tuned values are shown in the outermost right column. The optimized test score is about 92%.

symbol	description	range	optimal value
`NLL`	λ_r	$[10^{-1},10^{2}]$	0.784
`CE`	λ_c	$[10^{-1},10]$	1.403
`MMD`	λ_v	$[1,10^{3}]$	1.009
`LATENT_DIM`	latent variables dimension	[10,200]	16
`var_MMD`	σ in $\mathcal{N}(0,\sigma)$	[0.01,2]	0.646
`Learning Rate`	learning rate	[0.000 1, 1]	$6.6{\cdot}10^{-4}$

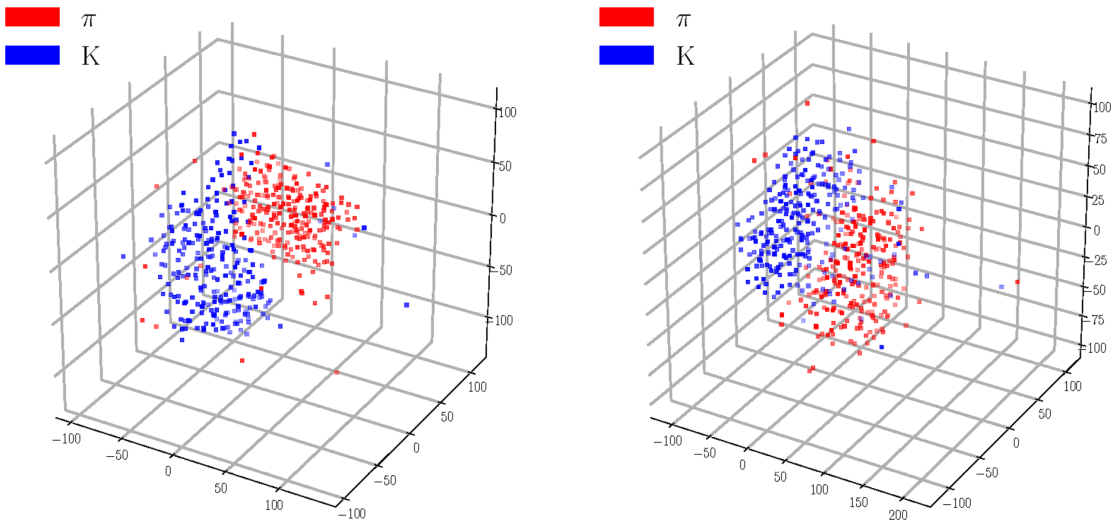

Figure 4. Example of features extracted by the CNN module from π's and K's at 4 GeV/c (left) and 5 GeV/c (right). These features are then used to classify the particle. The plot shows a better separation between π/K at 4 GeV/c, which means that the network has good distinguishing power. As expected the points become less separated at larger momentum. The 3D visualization is obtained with t-SNE [35]. Image taken from [23]

objective function f the development score obtained during the training phase. Each call of the BO is based on 50 epochs. Results of the optimization are summarized in table 2.

4. Results

The following results are based on charged π, K candidates with momentum between 4 and 5 GeV/c, the latter corresponding to a challenging kinematics given the sizeable overlap between the expected hit patterns. The capability of distinguishing πs from Ks and effectively doing PID depends on the features and the causal relations learnt in the space of the latent variables. A 3D visualization in the space of the latent variables is shown in figure 4, where t-SNE [35] is used for dimensionality reduction. A clearer separation is achieved in the reduced space of the latent variables at 4 GeV/c compared to 5 GeV/c.

An alternative representation of the same data is shown in figure 5. Here the distinguishing power is quantified as the average absolute difference between π and K in each latent variable versus the Y-position on the quartz bar. This is shown at 5 and 4 GeV/c in momentum (top and middle of figure 5, respectively). Notice that the number of bins (16 on the x-axis) corresponds to the dimension of the vector of latent variables. Intuitively, the larger the absolute difference the more πs are separated from Ks. The relative difference (bottom of figure 5) is characterized by negative values only, pointing to the obvious interpretation that the distinguishing power is larger at 4 GeV/c. Notice also that in good approximation the separation between the two particle types does not depend on the Y-location on the quartz bar and we verify as a sanity check the presence of vertical bars in the patterns of figure 5 along the y-axis.

As described in section 3.2, the event generation is based on FastDIRC which is also used in this section as a reconstruction algorithm to provide a benchmark against which evaluating the performance of the DeepRICH architecture.

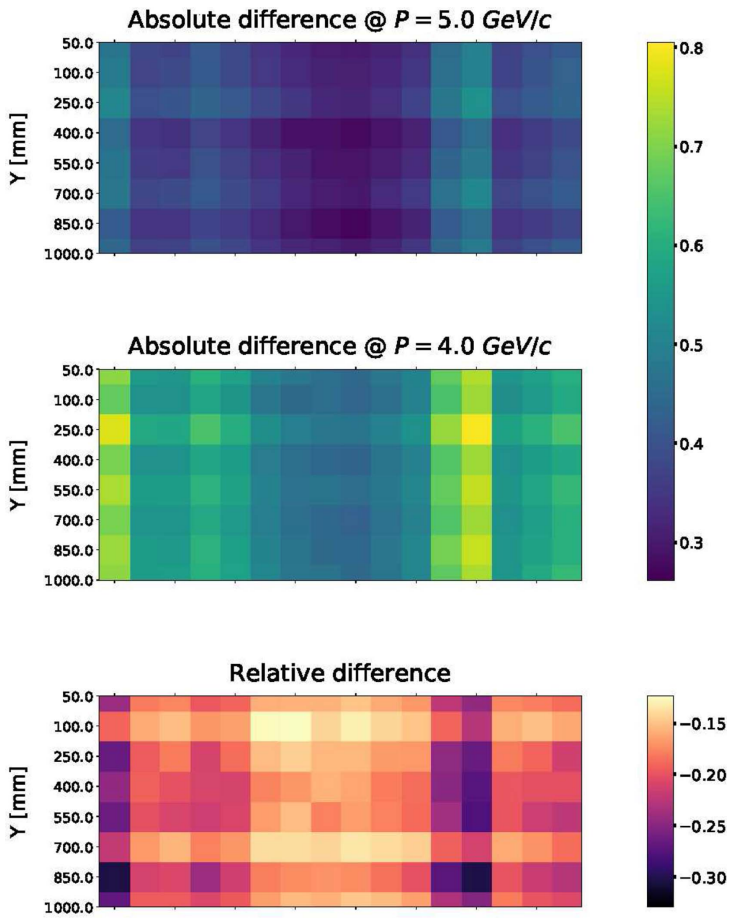

Figure 5. (Top and middle plots) 2D plot of the absolute difference on each latent variable between π's and K's, obtained for 5 GeV/c and 4 GeV/c, respectively. The latent variables are binned along the x-axis, whereas the y-axis shows where the particle traversed the bar (along Y [mm]). The color axis indicates the value of the absolute difference. The larger the difference the larger is the distinguishing power of the network. As expected the separation becomes less clear when the momentum is larger whereas there is no appreciable dependence on the position on the bar resulting in patterns with vertical bands. (Bottom) The relative difference between the first row (5 GeV/c) and the second row (4 GeV/c) showing negative values in the majority of the bins.

4.1. Comparison with fastDIRC

The PID strategy in FastDIRC is likelihood-based: N_d photons for each candidate particle are detected in the PMT plane, and N_g photons are generated to produce the expected PDFs of the 2 candidates (π, K). The N_d particles are then used to compute the log-likelihood from each candidate PDF as follows:

$$\log \mathcal{L}_{\pi(K)} = \sum_{j=1}^{N_d} \ln \Big(\sum_{i=1}^{N_g^{\pi(K)}} g\Big(\frac{|\mathbf{x}_i^{\pi(K)} - \mathbf{x}_j|}{\lambda}\Big)\Big), \tag{5}$$

where λ is a bandwidth and $\mathbf{x}$ is a vector whose components are the spatial and time coordinates of each hit (either detected or generated)[4].

The operational definition of likelihood in DeepRICH is different from equation (5), in that different quantities are provided by the network: as explained in section 3, the output of the classifier is a two-dimensional vector $\tilde{\mathbf{y}} \in \mathbb{R}^2$, and we use these values as likelihoods for π and K.

At this point we can consider the $\Delta \log \mathcal{L}$, the difference between the two log-likelihoods (under the null hypotheses of π and K, respectively). Histograms of $\Delta \log \mathcal{L}$ are obtained for both FastDIRC and DeepRICH and shown in figure 6 at 4 GeV/c (left column) and 5 GeV/c (right column), respectively. Two different colors are used in the legend to highlight the ground truth of each particle (which is either a real π or K).

In the same figures, to quantify the performance of the two algorithms, a Receiver Operating Characteristic (ROC) curve is obtained by changing the threshold on the $\Delta \log \mathcal{L}$ to cut on. The ROC curves

[4] In FastDIRC N_g is chosen such that the achieved resolution reaches a stable value, and the bandwidth is tuned to provide the best performance.

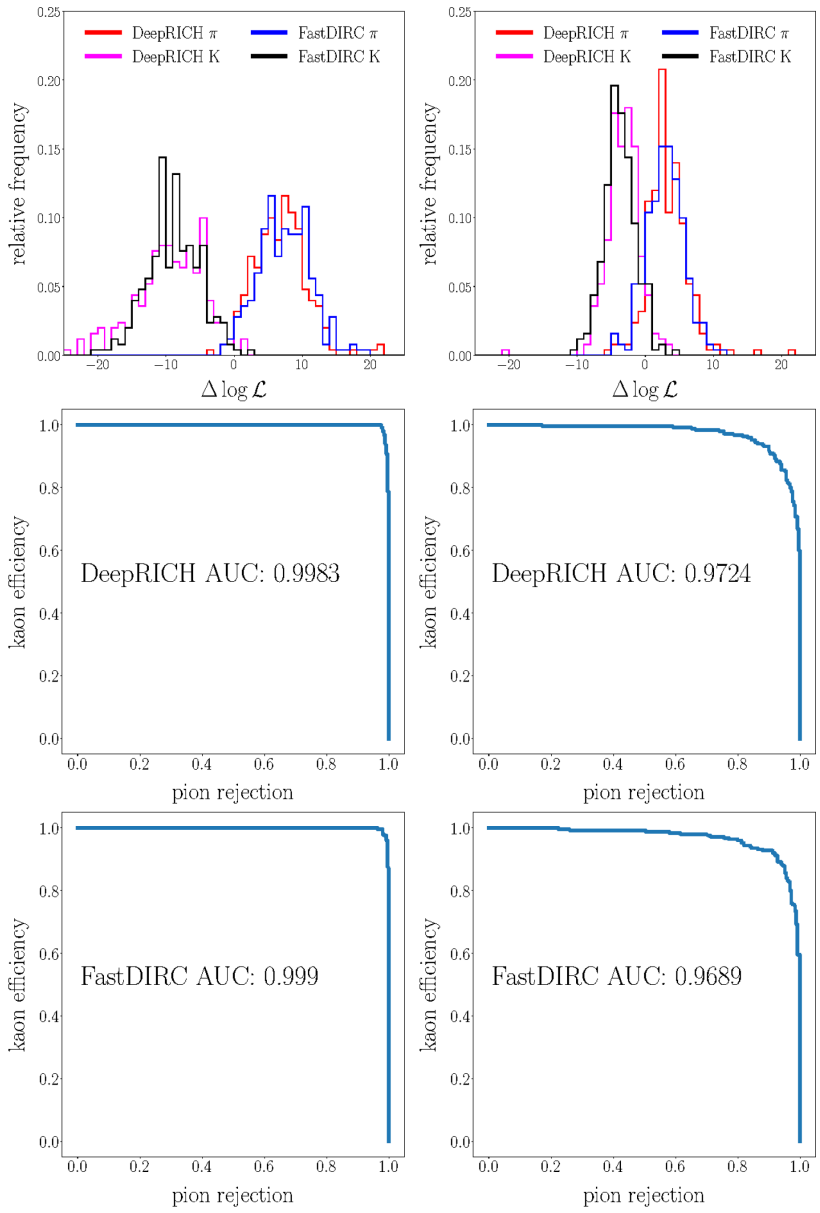

Figure 6. The results are obtained with $p = 4$ GeV/c (left column) and $p = 5$ GeV/c (right column), at $\theta = 2.0$ [deg], $\phi = 20$ [deg], X = -10.5 [cm], Y = 500 [cm] using 350 particles. (top row) The overlaid distributions of $\Delta \log \mathcal{L}$ (see text for definition) for DeepRICH and FastDIRC, from which the corresponding ROC curves are calculated for comparison: (middle) DeepRICH; (bottom) FastDIRC.

have been produced generating 350 particles observed for each kinematics and the Area Under Curve (AUC) is used as a metric to compare the performance of the two algorithms.

A detailed comparison between FastDIRC and DeepRICH reconstructions is reported in Figure 7 (top), where the DeepRICH AUC divided by the corresponding AUC of FastDIRC is drawn as a function of a single kinematic variable, after integrating the performance over all the other kinematic parameters to show the partial dependence on that particular variable.

The plots show that the two algorithms are very close in reconstruction performance, namely AUC(deepRICH) $\gtrsim 0.99 \cdot$ AUC(FastDIRC) in a large region of the kinematic parameters where the reconstruction efficiency of DeepRICH is approximately uniform, while a slight dependence is observed as a function of the momentum. Figure 7 (bottom) summarizes these results in form of radar plots: each axis correspond to a kinematic parameter, and the distance from the center on each direction corresponds to the correlation of the AUC with that specific parameter. As expected, the largest dependence of the AUC is on the momentum parameter, the π/K distinguishing power becoming lower at larger values of the momentum.

4.2. Test on unknown kinematics

One major concern about this method regards the predictability for kinematics not explicitly injected in the training phase. In this section we show results that prove the stability of the network reconstruction for every

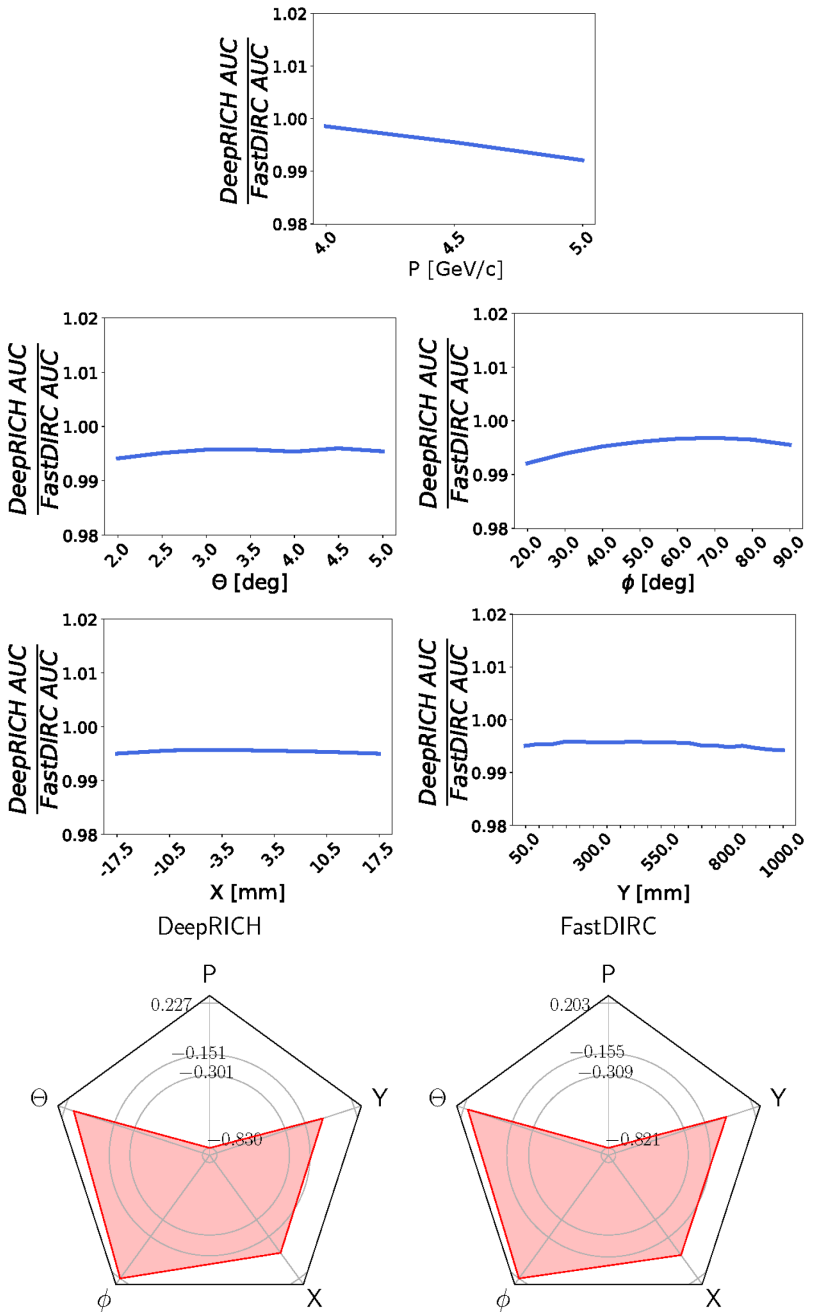

Figure 7. (top 3 rows) The ratio between DeepRICH and FastDIRC AUCs. Each AUC is calculated to show the partial dependence on one kinematic parameter by marginalizing on all other parameters. (bottom row) Radar plots representing the correlation between the AUC and each kinematics parameter for DeepRICH and FastDIRC, showing that the two reconstruction algorithms perform similarly as a function of the kinematic parameters. Notice at 4 GeV/c that the two reconstruction methods perform almost identically.

kinematic point belonging to the hypercube $\Delta p \times \Delta\theta \times \Delta\phi \times \Delta x \times \Delta y$, which was approximated in section 3.2 by a discrete grid of training datasets. This approximation tacitly assumes no discontinuities in the hit pattern by varying the parameters within the hypercube.

In figure 8 we show the quality of the DeepRICH reconstruction for unknown kinematics in terms of the test score. We performed different tests and we did not notice any sensible changes in the test score and in the AUC, which are two figures of merit we have used to prove the quality of the reconstruction.

4.3. DeepRICH performance

In this section we summarize the performance of the network both in terms of reconstruction efficiency and computing time.

The quality of the reconstruction is high as shown in table 3: as already mentioned, the AUC values are close to those of FastDIRC, given a certain sub-region of the kinematic space for the training process. Notice these results can further improve considering the major points addressed in sections 3.2–3.4 for the training phase.

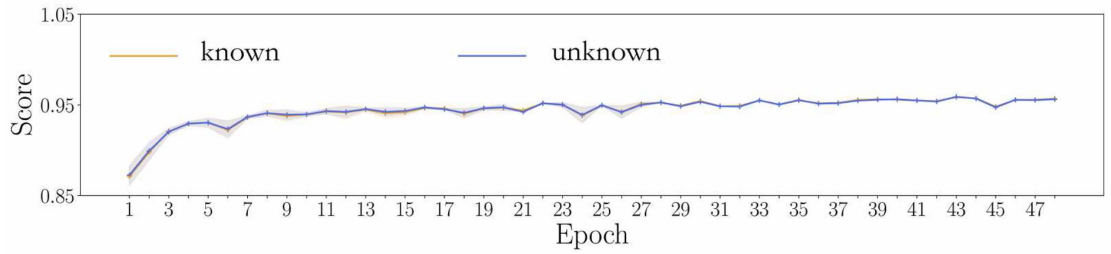

Figure 8. The image shows the learning curve corresponding to known and unknown kinematics combining the datasets with momentum ∈ [4,5] GeV/c. Each point is obtained as an average over 3 experiments—notice in some experiment the early stopping activated earlier. These results prove the ability of DeepRICH to reconstruct unknown kinematics.

Table 3. The area under curve (%), the signal efficiency to detect pions ε_S and the background rejection of kaons ε_B corresponding to the point of the ROC that maximizes the product $\varepsilon_S{}^*\varepsilon_B$. The corresponding momenta at which these values have been calculated are also reported. This table is obtained by integrating over all the other kinematic parameters (i.e. a total of ∼6k points with different θ, ϕ, X, Y for each momentum).

	DeepRICH			FastDIRC		
Kinematics	AUC	ε_S	ε_B	AUC	ε_S	ε_B
4 GeV/c	99.74	98.18	98.16	99.88	98.98	98.85
4.5 GeV/c	98.78	95.21	95.21	99.22	96.33	96.32
5 GeV/c	96.64	91.13	91.23	97.41	92.40	92.47

Table 4. Performance of the DeepRICH architecture, reporting the average inference time, the inference memory and the training memory, *i.e.* the GPU memory required by the network during the inference and training phases with a fixed batch size. The workstation uses a GPU Titan V with the CUDA10.0_0 build. The network has been implemented using PyTorch 1.2 [36].

specs	**value**
inference time per batch	$\mathcal{O}(1)$ ms
inference network memory	$\mathcal{O}(1)$ GB
training network memory	$\mathcal{O}(4)$ GB
network memory on local storage	∼ 6 MB
network trainable parameters	458 592

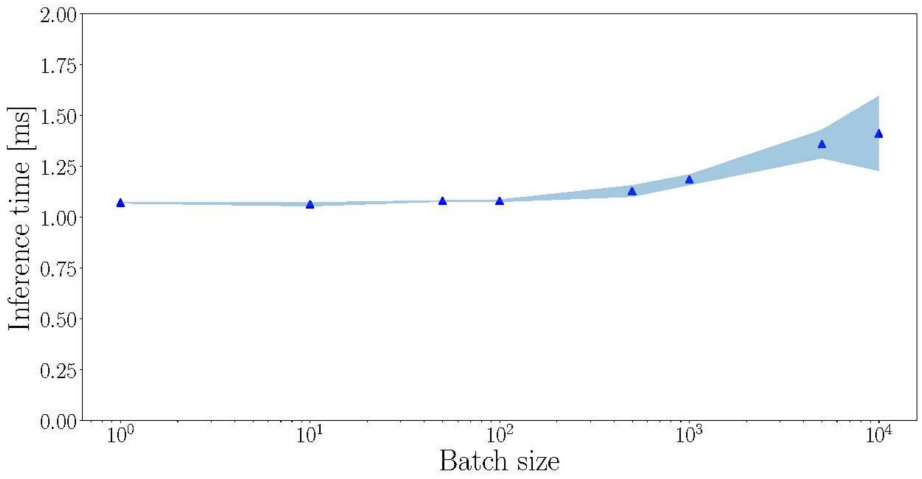

Figure 9. After training, the inference time is almost constant as a function of the batch size, meaning that the effective inference time—*i.e.*, the reconstruction time per particle—can be lower than a μs, the architecture being able to handle 10^4 particles in about 1.4 *ms* in the inference phase. Notice that the corresponding memory size in the inference phase is approximately equal to the value reported in table 4.

Table 4 reports on the memory and computing time performance: the inference time is the actual time DeepRICH needs to do PID after the training phase and is on average $\mathcal{O}$(ms) per batch of particles using a GPU Titan V. Figure 9 shows the inference time as a function of the batch size: the inference time is approximately constant up to 10^4 particles, which is the maximum batch size that could be handled in our configuration due to internal memory limits of Titan V in the inference phase.

For completeness we also report a comparison with the reconstruction time of other methods: for look-up-table-based algorithms, not fully optimized estimates provide order few ms per track on a single standard CPU [37]; for FastDIRC it is about 300 ms per track on a Macbook Air 2.2 GHz i7 and is dominated by the generation of the PDF, though it is worth reminding that can be massively parallelized; the GAN method [4] is the closest to our order of magnitude (but it regards the generation of $\Delta \log \mathcal{L}$ values) and the authors claim 1 M particles generated per second.

Another potential advantage of DeepRICH is the limited network size evaluated throughout all the training phase, which never exceeded 4 GB for different network configurations. It's worth reminding that the network size depends mainly on the weights of the network and the gradients, rather than on the subspace of the kinematic parameters used in the training phase.

This is a feature to keep in mind when comparing to the overall size of a look-up table obtained for example with the geometrical reconstruction method.

5. Summary and conclusions

The DeepRICH architecure developed in this paper shows very promising results. As a case study we consider the DIRC detector. Notice that DeepRICH is agnostic to the shape of the photon patterns, and in principle it can be trained to do PID for other imaging Cherenkov detectors.

The training set is generated with FastDIRC dividing the phase-space in a fine grid of points. We have made different tests changing the number of kinematic points in p, θ, ϕ, X, Y and for one specific bar of the DIRC (we refer the reader to section 3.2 for more details on the preparation of data).

We prove the high quality and stability of the reconstruction within the kinematic subspace. We then increased the space and kept the same dimensions of the neural network architecture, and this does not seem to affect the quality of the reconstruction. Notice that the generation of the hypercube and the resulting density can be further optimized in the future. Increasing the kinematic space and consequently the size of the dataset obviously results in larger training time and ideally this is limited only by computing resources and available time to train the network. The training time of DeepRICH has not been optimized. One can improve this in different ways, for example with a more sparse grid of points, or with distributed training, see, e.g. [38], However, without optimization, this time can be as large as half a day with the current configuration on a single Titan V GPU.

It is worth reminding that the size of the network is related to the weights and the dimensions of the architecture. The measured inference time is approximately equal to 1 ms per batch and we find it is roughly constant up to 10^4 particles. Notice that further parallelization of the network can be explored during both the training and inference phases.

Our conclusion is that DeepRICH, within the conditions described throughout the text, can reach the reconstruction efficiency of established algorithms and potentially outperform them in the reconstruction time.

The $\mathcal{O}$(ms) time performance per batch of 10^4 particles makes this algorithm suitable for near real-time applications (*e.g.* calibration). The high quality of reconstruction and the fast computing time are two compelling features of the DeepRICH algorithm, this coming at the cost of relatively long training time, as expected. If the latter aspect cannot be further optimized in the future, one can always use DeepRICH to characterize critical sub-regions of the phase-space, *e.g.* it can be applied to each bar separately.

DeepRICH has been designed to be easily generalized to classify other categories of particles, and the extension of the network is left for future development. An important feature is related to the nature of the VAE, which suggests a tempting scenario of generalizing DeepRICH to fast generation of events once the behavior in the latent space is learnt. Finally another suggestive application could be training DeepRICH using pure samples of identified particles from real data, this allowing to deeply learn the response of the Cherenkov detector.

Acknowledgments

We thank E Cisbani and M Williams for useful comments.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-FG02-94ER40 818.

CF was also supported by the research grant prize awarded by the Jefferson Lab Associates. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Data availability statement

The data that support the findings of this study are available upon request from the authors.

ORCID iDs

Cristiano Fanelli https://orcid.org/0000-0002-1985-1329
Jary Pomponi https://orcid.org/0000-0003-3236-3941

References

[1] Séguinot J and Ypsilantis T 1977 Photo-ionization and Cherenkov ring imaging *Nucl. Instrum. Methods* **142** 377–91
[2] Stevens J *et al* 2016 The GlueX DIRC project *J. Instrum.* **11** C07010
[3] Albertsson K *et al* 2018 Machine learning in high energy physics community white paper *J. Phys. Conf. Ser.* **1085** 022008
[4] Derkach D, Kazeev N, Ratnikov F, Ustyuzhanin A and Volokhova A 2020 Cherenkov detectors fast simulation using neural networks *Nucl. Instrum. Methods Phys. Res. A* **952** 161804
[5] Adam I *et al* 2005 The DIRC particle identification system for the BaBar experiment *Nucl. Instrum. Methods Phys. Res.* A **538** 281–357
[6] Va'vra J *et al* 2013 Progress on development of the new FDIRC PID detector *Nucl. Instrum. Methods Phys. Res.* A **718** 541–5
[7] Inami K and Belle I I 2014 TOP counter for particle identification at the Belle II experiment *Nucl. Instrum. Methods Phys. Res.* A **766** 5–8
[8] Charles M and Forty R 2011 TORCH: Time of flight identification with Cherenkov radiation *Nucl. Instrum. Methods Phys. Res., Sect.* A **639** 173–176
[9] Kalicy G *et al* 2016 High-performance DIRC detector for the future electron Ion collider experiment *J. Instrum.* **11** C07015
[10] Patsyuk M *et al* 2020 Status of the GlueX DIRC *Nucl. Instrum. Methods Phys. Res.* A **952** 161756
[11] Hardin J and Williams M 2016 FastDIRC: a fast Monte Carlo and reconstruction algorithm for DIRC detectors *J. Instrum.* **11** P10007
[12] Lacock P, Michael C, Boyle P and Rowland P 1997 Hybrid mesons from quenched QCD *Phys. Lett.* B **401** 308–12
[13] Meyer C A and Swanson E S 2015 Hybrid mesons *Prog. Part. Nucl. Phys* **82** 21–58
[14] The GEANT-based SuperB FDIRC Monte Carlo geometry has been implemented by Dzhygadlo R and Patsyuk M (Available at http://web-docs.gsi.de/ rdzhigad/www/)
[15] Dey B *et al* 2015 Design and performance of the Focusing DIRC detector *Nucl. Instrum. Methods Phys. Res.* A **775** 112–31
[16] Dzhygadlo R *et al* 2016 The PANDA Barrel DIRC *J. Instrum.* **11** C05013
[17] Dzhygadlo R *et al* 2014 Simulation and reconstruction of the PANDA Barrel DIRC *Nucl. Instrum. Methods Phys. Res.* A **766** 263–6
[18] Starič M, Inami K, Križan P and Iijima T 2008 Likelihood analysis of patterns in a time-of-propagation (TOP) counter *Nucl. Instrum. Methods Phys. Res.* A **595** 252–5
[19] Cranmer K 2001 Kernel estimation in high-energy physics *Comput. Phys. Commun.* **136** 198–207
[20] Goodfellow I J, Pouget-Abadie J, Mirza M, Bing X, Warde-Farley D, Ozair S, Courville A and Bengio Y 2014 Generative adversarial nets *Adv. Neural Inf. Process Syst.* **27** 2672–80
[21] Maevskiy A *et al* 2019 Fast Data-Driven Simulation of Cherenkov Detectors Using Generative Adversarial Networks *19th Int. Workshop on Advanced Computing and Analysis Techniques in Physics Research*
[22] Kingma D P and Welling M 2013 Auto-encoding variational Bayes *CoRR 2nd Int. Conf. on Learning Representations (ICLR) 2014*
[23] Fanelli C 2020 Machine learning for imaging Cherenkov detectors *J. Instrum.* **15** C02012
[24] Girshick R 2015 Fast R-CNN *IEEE Int. Conf. on Computer Vision (ICCV)* (ArXiv: 1504.08083)
[25] Gretton A, Borgwardt K M, Rasch M J, Schölkopf B and Smola A J 2007 A Kernel Method for the Two-Sample Problem *Adv. Neural Inf. Process Syst. 19 (NIPS 2006)* 513–520
[26] Zhao S, Song J and Ermon S 2017 InfoVAE: Information Maximizing Variational Autoencoders arXiv: 1706.02262
[27] Hoffman M D and Johnson M J 2016 Elbo surgery: yet another way to carve up the variational evidence lower bound *In Workshop in Advances in Approximate Bayesian Inference NIPS* p 1
[28] Schölkopf B 2001 The kernel trick for distances *Adv. Neural Inf. Process. Syst.* **13** 301–7
[29] Kingma D P and Jimmy B 2014 Adam: A Method for Stochastic Optimization arXiv:1412.6980 [cs.LG]
[30] Jones D R, Schonlau M and Welch W J 1998 Efficient global optimization of expensive black-box functions *J. Global Optim.* **13** 455–92
[31] Snoek J, Larochelle H and Adams R P 2012 Practical Bayesian optimization of machine learning algorithms *Adv. Neural Inf. Process. Syst.* **25** 2951–9
[32] Eggensperger K, Feurer M, Hutter F, Bergstra J, Snoek J, Hoos H and Leyton-Brown K 2013 Towardsan empirical foundation for assessing Bayesian optimization of hyperparameters *In NIPS workshop on Bayesian Optimization in Theory and Practice* vol 10 p 3
[33] Williams C K I and Rasmussen C E 2006 *Gaussian Processes for Machine Learning* (MA: 2.MIT press Cambridge)
[34] Head T *et al* 2018 scikit-optimize/scikit-optimize: v0.5.2 (Version v0.5.2) Zenodo (http://doi.org/10.5281/zenodo.1207017)
[35] van der Maaten L and Hinton G 2008 Visualizing data using t-SNE *J. Mach. Learn. Res.* **9** 2579–605
[36] Paszke A *et al* 2019 Pytorch: An Imperative Style, High-Performance Deep Learning Library *Adv. Neural Inf. Process. Syst.* pp 8024–35
[37] Dzhygadlo R 2019 *Private Communication*
[38] Shen Li 2017 PyTorch Model Parallel Tutorial (https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html)