university of
groningen

Bachelor of Physics

Undergraduate dissertation

Deep learning for particle tracking

Ignacio Fernández Graña

supevised by
dr. Xabier Cid Vidal
dr. Johan G. Messchendorp

Departamento de Física de Partículas da USC
&
KVI- Center for Advanced Radiation Technology at RUG

July 2020

To my family
and to those of you who made Groningen worth remembering.

Abstract

The main goal of this research project is to study the applications of a convolutional neural network (CNN) for identifying particle tracks in collider experiments, using inner tracking detectors such as those in the future PANDA experiment ($p\bar{p}$) and the running BESIII experiment (e^+e^-). This work is a follow-up of the master thesis by Harmjan de Vries who demonstrated the feasibility of the CNN methodology with tracking data generated by Monte Carlo simulations of e^+e^- collisions in the BESIII experiment. We tested the CNN designed in that thesis on events with more noise and more particle tracks, simulating high interaction rate conditions. The results show that the CNN is able to reduce background noise with high accuracy but does not efficiently filter particle tracks unrelated to the original event. Therefore, the applications of this CNN are restricted to events with low interaction rates without pile-up of other events.

Resumo

O obxectivo principal deste proxecto é estudar as aplicacións dunha rede neuronal convolucional (CNN) para a identificación de trazas de partículas en experimentos colisionadores, usando detectores internos como os dos experimentos PANDA ($p\bar{p}$) e BESIII (e^+e^-). Este traballo é unha continuación da tesis de máster de Harmjan de Vries, onde se demostrou a viabilidade da metodoloxía CNN usando datos xerados en simulacións Monte Carlo de colisións e^+e^- no experimento BESIII. Nós estudamos a CNN deseñada nesa tesis en eventos con máis ruído e máis partículas, simulando condicións de altas tasas de interacción. Os resultados amosan que a CNN é capaz de reducir o ruido cunha alta exactitude, pero non filtra eficientemente trazas de partículas non relacionadas co evento orixinal. Polo tanto, as aplicacións desta CNN quedarían restrinxidas a eventos con tasas de interacción baixas e sen solapamento doutros eventos.

Resumen

El objetivo principal de este proyecto es estudiar las aplicaciones de una red neuronal convolucional (CNN) para la identificación de trazas de partículas en experimentos colisionadores, usando detectores internos como los de los experimentos PANDA ($p\bar{p}$) y BESIII (e^+e^-). Este trabajo es una continuación de la tesis de máster de Harmjan de Vries, donde se demostró la viabilidad de la metodología CNN usando datos generados en simulaciones Monte Carlo de colisiones e^+e^- en el experimento BESIII. Nosotros estudiamos la CNN diseñada en esa tesis en eventos con más ruido y más partículas, simulando condiciones de altas tasas de interacción. Los resultados muestran que la CNN es capaz de reducir el ruido con una alta exactitud, pero no filtra eficientemente trazas de partículas no relacionadas con el evento original. Por tanto, las aplicaciones de esta CNN quedarían restringidas a eventos con tasas de interacción bajas y sin solapamiento de otros eventos.

Contents

1 Introduction **3**

2 Tracking detectors **4**

2.1 The BESIII detector 4

3 Deep learning and artificial neural networks **5**

3.1 Overview 5

3.2 Forward propagation 6

3.3 The learning process: backpropagation 7

3.4 The ADAM optimizer 8

3.5 Convolutional neural networks 9

3.5.1 Convolutional layers 9

3.5.2 Pooling layers 11

3.5.3 Batch normalization 11

3.6 Residual networks 12

3.7 Overfitting and underfitting 12

3.8 Programming frameworks 13

4 Network design **13**

4.1 Input processing and labeling 13

4.2 Output processing 14

4.3 Loss functions 15

4.4 Performance metrics 15

4.5 U-net architecture 16

5 Testing the model **17**

5.1 Testing on different background noise levels 18

5.2 Adding random tracks 21

5.3 Testing on incomplete events 23

6 Conclusions **24**

1 Introduction

The upcoming PANDA experiment, acronym which stands for antiProton ANnihilation at DArmstadt, will be on of the main experiments at the Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany. It will investigate a wide range of fundamental questions in hadron and nuclear physics by studying collisions of antiprotons and fixed nucleons or nuclei in a momentum range of 1.5-15 GeV/c. Some of the fundamental topics which will be studied in PANDA are gluonic excitations, nucleon structure and the physics of exotic hadrons, among others [2].

Modern experiments in particle physics like PANDA commonly generate large amounts of data due to the high number of particle interactions, making the data storage and analysis a rather challenging computational task. The traditional approach when analyzing this data assumes that the events of interest can be efficiently selected in real time, and once selected, they can be affordably distributed and stored for further off-line analysis. Nevertheless, in modern experiments these assumptions break down as the amount of data produced is rapidly increasing. In particular, in PANDA the interaction rates will be as high as $2 \cdot 10^7$ interactions per second with typical event sizes of 4-20 kB, leading to data rates around 200 GB/s [3]. The imposed storage capacity limitations ($\approx$ 3 PB/year) do not allow to store all the generated data, meaning that the data rates need to somehow be reduced by three orders of magnitudes to match these capacity limitations. To do so, a real time event selection algorithm which reconstructs the particle tracks and discriminates the events of interest is needed. A potential candidate for an algorithm of such characteristics is a machine learning algorithm; specifically in this project we will study an artificial neural network.

Machine learning comprises any algorithm that is able to improve through experience and to learn from data. In recent years, it has proven to be an extremely useful tool in many fields, in particular to analyze and extract features from large amount of data [4]. These algorithms are widely used in big companies such as Google, Amazon or Facebook, as well as in many fields of science. Deep learning is a subfield of machine learning whose central topic are a kind of algorithms loosely based on the biological functioning of the human brain: artificial neural networks, simply referred as neural networks (NN). NNs have been shown to perform very well in a wide variety of tasks like object detection and classification, speech recognition and even natural language processing tasks such as sentiment analysis or language translation [15]. Particularly, in areas related with image analysis a special class of deep learning algorithms called Convolutional Neural Networks (CNN) has been applied with outstanding results [17, 10]. Since particle tracking detectors are usually imaging detectors, CNN algorithms are very well suited for analysing the data produced in such detectors. The architecture of the CNN studied in this project is based on the U-Net network, a very popular CNN within the medical imaging community [10, 18].

In this project we will study a CNN as a potential candidate for the analysis of data from tracking detectors such as PANDA, as a follow-up of the master thesis written by Harmjan de Vries: *Convolutional Neural Network for Reducing Noise and Detecting Tracks in the BES-III Main Drift Chamber* [1]. In this thesis an U-Net based CNN was designed and it was trained as two different algorithms, a noise reduction algorithm to reduce background noise and a track recognition algorithm to identify the different particle tracks in the events. The data used to train and test the CNN were taken from Monte Carlo simulations in the Beijing Spectrometer III (BESIII), which studies e^+e^- collisions in an energy range of 2-4.7 GeV. Currently running experiments such as BESIII work at lower interaction rates than future experiments like PANDA, where the high interaction rates will cause overlap between different events. Therefore, BESIII generates much cleaner events with less background noise and less particles. In [1], both the noise reduction and the track recognition algorithms were shown to work very efficiently in these clean data. The main goal of this project is to test the limits of the applications of the CNN in data with more noise and more particle tracks, where high interaction rate conditions are simulated. Given that most running and future experiments use similar tracking detectors such as the ones in BESIII and PANDA, the results

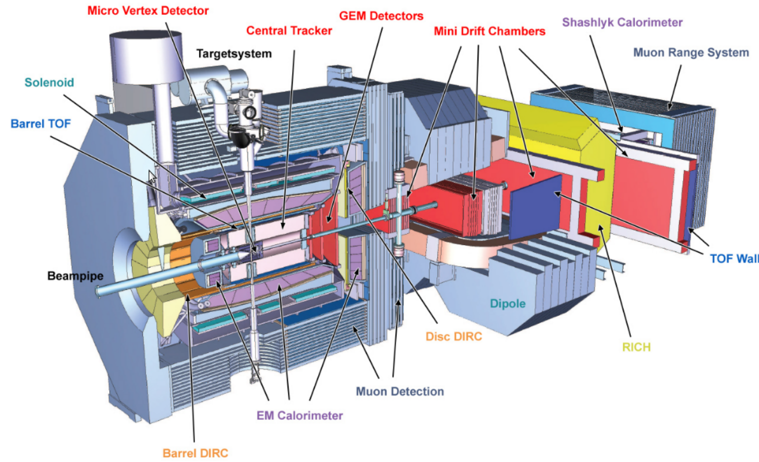
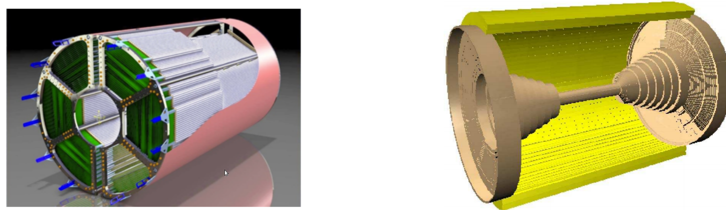


Figure 1: Layout of the PANDA detector. Image taken from [2].



(a) Straw Tube Tracker in PANDA (b) Main Drift Chamber in BESIII

Figure 2: Main tracking detectors in both the PANDA (Straw Tube Tracker) and the BESIII experiments (Main Drift Chamber). Image (a) taken from [6] and image (b) taken from [7].

obtained in this project can be generalized to many other experiments.

2 Tracking detectors

Both the PANDA and the BESIII experiments use similar tracking detectors to detect charged particles. The trajectory of a particle is measured by wires surrounding the interaction point. When a particle hits one of the wires, the position and time of the hit are stored, allowing to reconstruct the particle track. Unfortunately, noise coming from many sources is also detected in the wires. To measure the momentum of the charged particles, a magnetic field is applied through the detector. The curvature of the trajectory of a charged particle through the magnetic field depends on the momentum of the particle. Thus, the momentum of the particle can be measured if the particle track can be properly identified and reconstructed. In the case of PANDA, the main tracking detector for charged particles will be the Straw Tube Tracker (STT) [6, 2]. It will contain 4636 aluminised tubes, called straws, disposed in 27 layers around the beam-target interaction point, and placed in a powerful 2 T magnetic field (see figures 1 and 2).

2.1 The BESIII detector

Artificial neural networks need labeled data to be trained and tested. For this purpose we will use data generated in Monte Carlo simulations from the Beijing Spectometer III (BESIII) experiment, which has

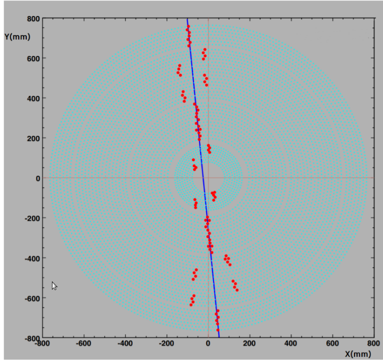


Figure 3: Example of an event from the Main Drift Chamber in BESIII where the arrangement of the wire is shown. The blue line is a true particle track from cosmic rays going through the detector and the red dots are the measured hits in the wires. For the tilted wires, the measured hits do not correspond to the true particle position. Image taken from [8].

an inner detector similar to that of PANDA. This experiment studies e^+e^- collisions in a energy range of 2-4.7 GeV. The interaction rates in BESIII are lower than in PANDA, which translates in cleaner events with less background noise and less tracks. The noise data we will use is taken from real data that were triggered on random timing signatures, not related to the e^+e^- interaction point. With the available Monte Carlo data and noise data we can simulate high event rate conditions using mixing of events.

The BESIII innermost tracker is the Main Drift Chamber (MCD), containing 6796 signal wires arranged in 43 cylindrical layers coaxial to the beam pipe [7]. These layers can be axial layers, containing wires parallel to the beam direction, or stereo layers, containing tilted wires with a small angle with respect to axial direction. The wires are arranged, in an outer radial direction, in 8 stereo layers, 12 axial layers, 16 stereo layers and 7 axial layers. In this case, a superconducting solenoid magnet will provide an axial 1 T magnetic field throughout the tracking volume to curve the path of the particles. The main layout of the MDC in BESIII is very similar to that of the STT in PANDA. The measured hits in the wires can be stored in a 2D image representing the transverse section of the beam pipe, as seen in figure 3.

The data used in this project corresponds to the decay channel $e^+e^- \to \Psi(2S) \to J/\Psi\ \pi^+\pi^- \to e^-e^+\pi^+\pi^-$. The $\Psi(2S)$ and J/Ψ are both excited states of the charmonium system $c\bar{c}$, a system composed of a charm quark and a charm antiquark and bounded by the strong interaction. Charmonium is a powerful tool for the understanding of the strong interaction. Thanks to the high mass of the charm quark ($M_c \approx 1.5\mathrm{GeV/c^2}$), the dynamics of the bounded system $c\bar{c}$ can be approximated in a non-relativistic approach. Some states in the charmonium spectrum are still not well measured and are a central topic in the physics program of PANDA and BESIII, where further studies about the charm spectrum will be conducted.

3 Deep learning and artificial neural networks

3.1 Overview

Machine learning and in particular deep learning has changed the paradigms of computation in recent years. This revolution has been driven by two main reasons:

- First, the very large amount of data generated by society nowadays has created a need for tools to analyze this data, where machine learning algorithms have proven to be tremendously powerful [12].

- Second, the rapid increase in both computing power and memory capacity in the past few decades allowed the design of state of the art machine learning algorithms that can process and analyze very large amounts of data.

Machine learning and all its different sub-classes of algorithms are now a widely used tool in both industry and science. One of these sub-classes is deep learning, whose central topic are artificial neural networks, simply referred as neural networks (NNs). The functioning of a NN is loosely based in the human brain, as it consists of many connected units called neurons which are connected in a similar way to that of the neurons in a human brain. Each neuron can be seen as a processor which receives an input and computes, through some internal parameters, an output. A neural network is able to learn from the data the correct values of these internal parameters to execute a given task. Neurons are disposed in layers; the basic structure of a NN is composed of an input layer, one or more hidden layers and an output layer (see figure 4). 'Deep' neural networks have, by definition, more than one hidden layers, in contrast with 'shallow' neural networks that only have one or none hidden layers. In deep-learning networks, each layer of neurons trains on a distinct set of features based on the previous layer's output. The further you advance into the neural net, the more complex the features your layers can recognize, since they aggregate and recombine features from the previous layer. This allows the network to learn highly non-linear functions that otherwise would be impossible.

Artificial neural networks are a type of what is called supervised learning algorithms, where 'labeled' samples are presented to the network. These labeled samples contain the ground truth, i.e., what we want the network to predict, allowing the network to compare its predictions with that ground truth in order to learn. In this learning process, information needs to flow in two opposite directions. In forward propagation, the network is fed with the raw data through the input layer, going through all the hidden layers and ending up in the output layer, where the prediction is made. In addition, the information also goes backwards, from the output layers to the input layer, allowing the network to learn from its previous predictions. This process is called backpropagation.

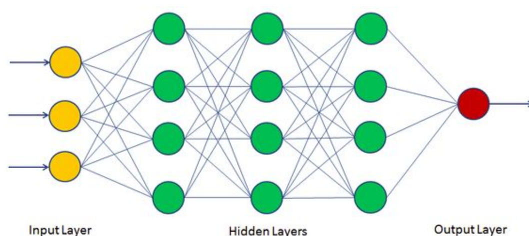


Figure 4: The scheme of a neural network. Deep neural networks have many hidden layers in between the input and the output layers, in contrast with shallow neural networks.

3.2 Forward propagation

During forward propagation the input data flows from the input layer all the way to the output layer going through every neuron in every layer. Neurons receive an input and compute an output through two operations: a linear function called *net function* and a non-linear function called *activation function.* The net function is a weighted average of all the inputs x_i, given by the equation:

$$z = \sum_{i=1} w_i \cdot x_i + b. \quad (1)$$

Here z is the net function, w_i are the weights, b is the bias and i indexes all the input values. In a fully connected layer, i would index all the neurons in the previous layers. The parameters w_i and b are the

learnable parameters of the neuron. The network, through the learning process, optimizes these values to make better predictions.

At a first glance, the activation function can look unnecessary. Nevertheless, without an activation function, our neural network would just be a linear combination of input values, resulting in a linear function itself. There are several activation functions commonly used, but in this work we will restrict ourselves to two of them:

- Rectified linear function (ReLU):
$$\mathrm{ReLU(z)} = \max(0, \mathrm{z}). \tag{2}$$
ReLU activation functions in the hidden layers have shown to yield better performance than hyperbolic or sigmoid activation functions in supervised training of very deep neural networks, in spite of the hard non-linearity and non-differentiability at zero [13]. In the NN studied in this project all the layers except from the output layer use ReLU as the activation function.

- Sigmoid function:
$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{3}$$
The sigmoid function is useful when we need the output of a neuron to be a value in between 0 and 1. For every point in the input image, our network will predict an output value which corresponds to the probability of that point of being a track point, so all the output values should be between 0 and 1. For that reason we will use a sigmoid activation function in the output layer.

The neurons in each layer will output a value after passing the operations mentioned before. The output of a layer will be the input of the next one, until the output layer is reached. In the output layer, the network will output a final prediction and it will calculate a function called *loss function* to measure how far from the ground truth (the label) the prediction is. The goal of the neural network is to minimize the difference between the prediction and the label of each input sample. To measure how well is the network doing in the whole training set, we need to take in account the loss function computed with all the samples of the training set. We do that with the so-called *cost function*. During backpropagation the network will try to minimize this cost function to learn how to make better predictions. The cost function is usually an average of the loss function taken over a specific number of samples m from the training set:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^i, y^i), \tag{4}$$

where $\hat{y}^i$ is the label or ground truth, y^i is the network's prediction and $L(\hat{y}^i, y^i)$ is the cost function for the input sample i.

3.3 The learning process: backpropagation

The network minimizes the cost function through the *optimization function* or *optimizer*. The goal of optimizer is to find the set of internal parameters in the network which minimizes the cost function. There are many optimization functions one can use in a neural network, but most of them are based on the gradient descent method [15]. Gradient descent is a way to minimize an objective function, in this case the cost function $J(w, b)$ by updating the parameters in the direction of the gradient of the objective function $\nabla J(w, b)$. The learning rate, η, determines the size of the steps we take to reach a the minimum. In other

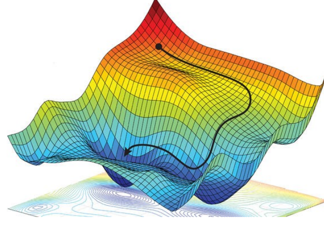


Figure 5: Abstract representation of a multivariable function where the path the optimizer follows to get to the global minimum is painted in black. The cost function is a highly dimensional function that depends on all the internal parameters of the network. The CNN studied in this project has around 5 millions parameters.

words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley, as shown in figure 5.

Depending on the number m of training samples we take in account in the cost function (equation 4) before updating the internal parameters of the network, we can have three different training protocols:

- In **batch training**, also know as vanilla gradient descent, the whole training set is used to compute the gradients, so the parameters are updated only once every time the network goes trough the whole training set. This pass through all the data from the training set once is called an epoch.
- In **mini-batch training** the training set is divided in batches, with the parameters being updated after each batch is presented to the network. This is useful, for instance, when the whole training set does not fit in memory. In addition, this protocol usually converges more rapidly.
- An extreme case of mini batch training would be **stochastic training**, where the parameters are updated after each sample in the training data set, i.e., batches have size 1.

Batch gradient descent can be slow as it needs to calculate the gradients of the whole training set in order to perform one update. On the other hand, when using stochastic training it can be difficult to find the exact minimum of the cost function as the network might keep iterating without reaching the minimum. To avoid these problems a mini-batch size of 50 was used in this project, as it was the chosen size in the master project [1].

3.4 The ADAM optimizer

As we mentioned earlier, most optimizers are based on the gradient descent method, which allows to find the global minimum in a multivariable function, in this case the cost function $J(w, b)$. The ADAM optimizer (ADAptive Momentum), first proposed in [24], is an upgrade of the classical gradient descent optimizer and it is the chosen optimizer for our network, as it has been shown to be an overall good choice as an optimizer in a wide range of cases [15, 16].

The classical gradient descent optimizer uses a fixed learning rate which does not change during training. Instead, ADAM is an adaptative learning rate method, which means it computes individual learning rates for different parameters; this allows to drive the convergence of the algorithm more efficiently towards the minimum. It does so by taking in account two different momenta, m and v.

The specific operations used to update the internal parameters of the network, w and b, are:

$$\Delta w_{ij} = -\eta \frac{m_{ij}}{\sqrt{v_{ij}} + \epsilon} , \qquad \Delta b_{ij} = -\eta \frac{m_{ij}^{bias}}{\sqrt{v_{ij}^{bias}} + \epsilon} , \tag{5}$$

where m_{ij} and v_{ij} are the first and second momentum, respectively, η is the learning rate and ϵ is an added parameter for numerical stability. The momentum terms for the weights w_{ij} and for the bias term b_{ij} are computed as:

$$
\begin{aligned}
m_{ij} &= \frac{\beta_1 \bar{m}_{ij} + (1-\beta_1)\frac{\partial J}{\partial w_{ij}}}{1-\beta_1^t}; \qquad v_{ij} = \frac{\beta_2 \bar{v}_{ij} + (1-\beta_2)(\frac{\partial J}{\partial w_{ij}})^2}{1-\beta_2^t}; \\
m_{ij}^{(bias)} &= \frac{\beta_1 \bar{m}_{ij}^{(bias)} + (1-\beta_1)\frac{\partial J}{\partial w_{ij}}}{1-\beta_1^t}; \qquad v_{ij}^{(bias)} = \frac{\beta_2 \bar{v}_{ij}^{(bias)} + (1-\beta_2)(\frac{\partial J}{\partial w_{ij}})^2}{1-\beta_2^t},
\end{aligned}
\tag{6}
$$

where $\bar{m}_{ij}$ and $\bar{v}_{ij}$ are the respective momenta calculated in the previous step, and t is the number of steps taken so far. For the parameters β_1 and β_2 in equation 6 as well as ϵ in equation 5, we have taken the default values from the original paper: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [24]. The learning rate used is $\eta = 0.001$, chosen based on the study done in [1] where it was shown to yield the fastest learning

With the momentum terms, the ADAM optimizer takes into account not only the gradient of the cost function for the current step, but also for all the previous steps. The first and the second momentum terms, m and v, are respectively exponentially moving averages of the first derivative and second derivative of the cost function with respect of the corresponding internal parameter: $\frac{\partial J}{\partial w_{ij}}$ and $(\frac{\partial J}{\partial w_{ij}})^2$ in the case of the weights w_{ij} and $\frac{\partial J}{\partial b_{ij}}$ and $(\frac{\partial J}{\partial b_{ij}})^2$ in the case of the bias term. An exponentially moving average is a weighted average where the weights of the values calculated in past steps decrease exponentially as we go further in the past. Multiplying the learning rate by the first momentum in equation 5 increases the size of the steps taken in a direction that coincides with the average direction of the previous steps. This happens because the first momentum increases when the previous steps have been taken in similar directions, much like the classical momentum of a particle [16]. In the same way, dividing the learning rate by the square root of the second momentum decreases the step size when the gradients are varying too much, i.e., the square of the gradient is big. In this case, the oscillations do not cancel out because the squares are always positive, so this term gets larger whenever the previous steps have oscillated a lot.

3.5 Convolutional neural networks

In general, the standard architecture of neural networks is the one presented in figure 4. It consists of an input layer, an output layer and a given amount of fully connected hidden layers in between, meaning that every neuron in a given layer receives input from every neuron in the previous layer. For tasks related with computer vision, like pattern recognition or image analysis, there is a special type of neural network that is known to outperform the standard NN architecture [12]: *convolutional neural networks* (CNN). The origin of the popularity of this kind of architecture was the design of the CNN 'AlexNet' for the ImageNet competition in 2012 [25], which exhibited an unprecedented low error in object recognition. The architecture of a CNN is more diverse as they can be made up from three types of layers: convolutional, fully connected and pooling layers. A convolutional network which does not use any fully connected layers is called a fully convolutional network.

3.5.1 Convolutional layers

The core building block of a convolutional neural network is the convolutional layer, which uses a special type of linear operation between arrays called convolution. The layer's learnable parameters are encoded in the filters (or kernels), 2-dimensional arrays (could be arrays of higher dimensions but in this project we will only use 2-dimensional filters). Each filter can be seen as matrix containing $n_f \times n_f$ weights and a bias term that act on input matrices (images) with size $n_{input} \times n_{input}$. During forward propagation,

each filter is 'convolved' accross the input matrices, meaning that the dot product between the filter and each $n_f \times n_f$ sub-matrix of the input images is computed and the bias term is added. The output, also called the feature map, is an image where each element is the result of each dot product (see figure 6). The dot product here is defined as multiplying element-wise the two matrices and summing all the values of the resulting matrix, so the result is a scalar.

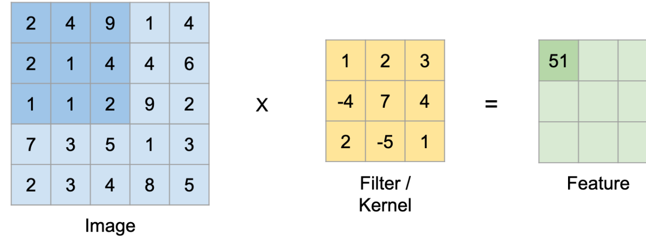


Figure 6: The convolution operation for the first 3x3 sub-matrix in the input image, without considering the bias term. The same operation should be repeated for all the 3x3 sub-matrices in the input image.

A convolutional layer can have one or more filters. If there is more than one, each filter is applied independently to the input image, so the output is a volume constructed by stacking up all the individual features maps (see figure 7). The depth (also referred as the number of channels n_c) of the output volume will be equal to the number of filters in the layer. There can also be the case where the input image has more than one channel (for instance, photographies usually have 3 channels, one for each basic color: red, blue and green). In this case, filters must have the same number of channels than the input image, and the convolution is also taken across the third dimension (see figure 7). Let's assume the input of a convolutional layer is an array with dimensions $n_{input} \times n_{input} \times n_c$. If we apply to this image m filters with size $n_f \times n_f$, then the output feature map will have a size $n_{output} \times n_{ouput} \times m$ where n_{output} is given by $n_{ouput} = n_{input} - n_f + 1$. We assume that the input images are squared, which will always be the case in this project.

Padding and stride

The size of an image shrinks when passing through a convolutional layer. To avoid this, a parameter p called padding can be additionally considered. When padding is applied, the size of the input image is increased by adding p columns and p rows to it, making the output image bigger too. For instance padding p=1 increases the size of the input image from $(n_{input}, n_{input}, n_c)$ to $(n_{input}+2, n_{input}+2, n_c)$. In this project we will only work with zero-padding, in which all the values of the added columns and rows are zero. When 'same' padding is used, p is chosen to be such that the size of the output image and the input image are the same. Padding also helps to better analyse the information located in the corner of the input image, since more convolution operations reach the values in the edges.

Another parameter in a convolutional layer is the stride s. A convolution operation with a stride not equal to one does not apply the filter on all possible sub-matrices in the input image, moving over the images in steps of s instead. This reduces the size of the the output image by a factor of s. After applying a stride s and a padding p, the width and height of the output image, n_{output}, will be given by:

$$n_{output} = \frac{n_{input} + 2p - f}{s} + 1. \tag{7}$$

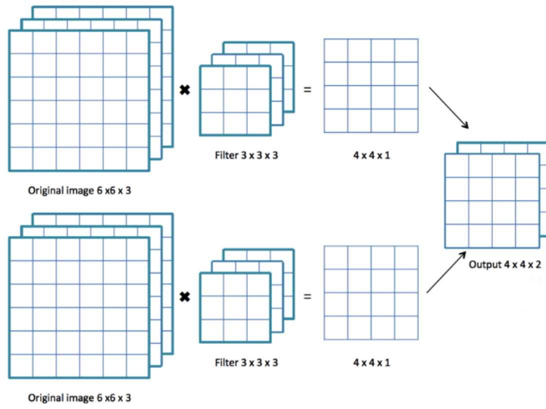


Figure 7: A convolutional layer where two filtres are applied to a $6 \times 6 \times 3$ input image. Each of the filters is individually applied to the input image generating two $4 \times 4 \times 1$ different feature maps. These two feauture maps are then stacked up to make a final $4 \times 4 \times 2$ output image.

3.5.2 Pooling layers

Convolutional networks often apply pooling layers in-between convolutional layers. The most common type of pooling, and the one used in this project, is *max-pooling*. Max-pooling divides the input image into a set of nxn non-overlapping sub-regions and, for each such sub-region, outputs the maximum. There are other pooling techniques such as average-pooling, median-pooling or min-pooling, where instead of taking the maximum of each subregion we take the average, the median or the minimum, respectively. When using a nxn pooling, the size of the input image will be reduced by a factor of $2n$, hence reducing the amount of parameters and computation in the network.

3.5.3 Batch normalization

Another technique widely used in deep learning is *batch normalization*, first proposed in [21]. It is shown to increase the speed and the stability of the learning process and leads to a better overall convergence. Moreover, it can even work as a normalization technique, helping to reduce overfitting. The reasons behind its effectiveness are still not well understood. Although it was thought to be because it reduces the internal covariant shift in the networks, later studies like [22] have contradicted this belief, while still not giving a definite answer.

Batch normalization works by normalizing all net activation values in a layer, substracting the mean μ and dividing by the square of the standard deviation σ^2. For a convolutional neural network, we would need to do this for every mini-batch of samples b, in every channel of the sample c and for all the pixels in the channel. The net activation after batch normalization would be:

$$z^{norm}_{b,c,x,y} = \frac{z_{b,c,x,y} - \mu_c}{\sqrt{\sigma^2 - \epsilon}} \quad , \quad \forall b, c, i, j \tag{8}$$

where $z_{b,c,i,j}$ is the net activation of a pixel located in the (i, j) position in the channel c of a sample from the mini-batch b. Through this process, the distribution of z^{norm} is set to have $\mu = 0$ and $\sigma^2 = 1$. This normalization can lead to a decrease in the 'expressive power' of the network [23], limiting the learning process. To avoid this, it is common to add two learnable parameters γ, β which allow the network to scale the parameters of the distribution in the following way:

$$y_{b,c,x,y} = \gamma_c \, z^{norm}_{b,c,x,y} + \beta_c, \tag{9}$$

where now the mean γ_c and the standard deviation β_c of the distribution can be learn by the network through the training process.

3.6 Residual networks

Very deep neural networks, with many layers of neurons and thus many internal parameters, are proven to perform really well, being able to learn very complex features from data. Nevertheless, one cannot endlessly stack more layers in a network in order to make it more accurate. At some point, if too many layers are added, accuracy reaches a maximum and then it may start decreasing. This 'depth problem' can be solved through the use of *residual networks* (ResNets), first proposed by Hel et al. [20], which implement the so-called skip connections. With the use of these skip connections, the output of a layer is not only used as the input for the next one, but it will also be the input of some other layer which is deeper in the network, as it was taking a shortcut (see figure 8). One reason behind the effectiveness of this skip connections is that they allow the network to easily learn the identity function, as in very deep neural networks this identity function can be sometimes difficult to learn due to the very large number of parameters. ResNets allow the information to skip certain layers which might be not needed improving the flexibility of the network. The CNN used in this project incorporates the use of skip connections

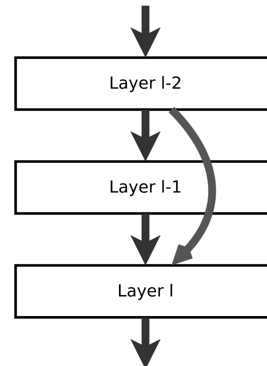


Figure 8: Scheme where the flow of information in a residual network is shown. Black arrows represent the standard way information flows, from one layer to the next one; the black arrow shows the way skip connections connect to non-consecutives layers in the network.

3.7 Overfitting and underfitting

When training a NN, it is common to not use all the data available to train the network. Instead, a small fraction is used to have an unbiased estimate of the performance of the network in data the network has never seen. Thus, the available data is splitted into two sets; a training set containing the data in which we train the network and a test set used to measure performance. The test set must be a good representation of the data set as whole, so it must be large enough to yield statistically meaningful results, while keeping in mind that we should train the network with as much data as possible for optimal performance.

In the training process of a neural network the goal is to learn the mathematical mapping between the input values and the desired outputs. This mapping can be really complex, specially in deep neural networks, and in general we cannot have an insight on the operations the neural network computes; it is a so-called black box model. Generalizing this mapping from the training set to the test set is not always easy. Sometimes the network might perform very well on the training set but has a much big error in the test set; this is called *overfitting.* Overfitting is usually due to two main reasons: the first one is that the data used in the training set is not enough or it does not have enough quality, and the second one is that the model we built is too complex for the given problem. In both cases, the network tends to memorize the data instead of learning it, meaning that is not able to generalize performance to data it has never seen. Techniques that help the network to overcome overfitting and to generalize better are known as

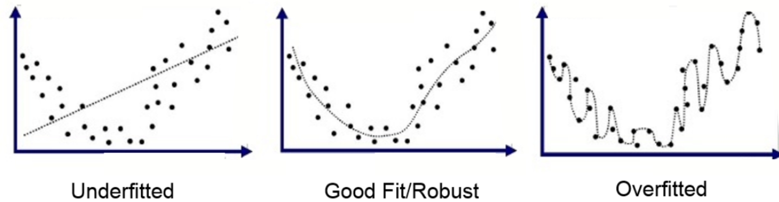


Figure 9: Schematic representation of an underfitted model, a well fitted model and a overfitted model.

regularization techniques. There are many regularization techniques one can apply to the network such as L1 and L2 regularizations, data agumentation, early stopping or dropout [11].

It could also be the case where the network is not able to perform well in any of the sets, neither the training nor the test set. This is called *underfitting*, and it is normally caused by a network that is not big and deep enough, i.e., it does not have enough parameters to learn the underlying mapping between the input data and the output. Identifying which problem does our network present is very important in order to try to solve it with the proper techniques.

3.8 Programming frameworks

The convolutional neural networks used in this project are deep networks with around 5 million parameters. Building a NN with these characteristics from scratch is not feasible. Luckily, there are many programming frameworks for machine and deep learning which facilitate the design and development of deep learning models, such as Caffe from UC Berkeley, CNTK from Microsoft, TensorFlow from Google, Torch and many other tools like Theano. In this project we are using Keras, a deep learning API written in Python, running on top of TensorFlow. Although Keras stands out for its easy and fast implementation, it is worth mention that other frameworks such as Caffe or PyTorch tend to be faster [14].

About the used hardware, we run the scripts in a 48 multi-core CPU machine. In general, the performance of the previous mentioned frameworks, including Keras, is much better on GPU systems than on many-cores CPU machines. Therefore, the time used by the network to analyze the images could be potentially improved by using other deep learning frameworks and by running them on a GPU machine.

4 Network design

The main goal of this project is to test the limits of the applications of the CNN designed in the master thesis [1] in events with high interaction rates conditions. Although two different algorithms were designed (noise reduction and track recognition), they are both based on the same CNN architecture, with slight changes in each of them. In this section we will summarize the insights of the network design that were previously obtained.

4.1 Input processing and labeling

The data we receive from the tracking software are the X and Y coordinates of the points of the wire where a particle is measured. For the tilted wires, the coordinates in the XY plane are taken as the average of the coordinates corresponding to the two ends of each wire. The raw data need to be shaped in a specific way to be used as input for the network, in a process called binning. In the binning process the values of the X and Y coordinates of a event are stored in a 2D image with a pixel resolution of 192x192. This

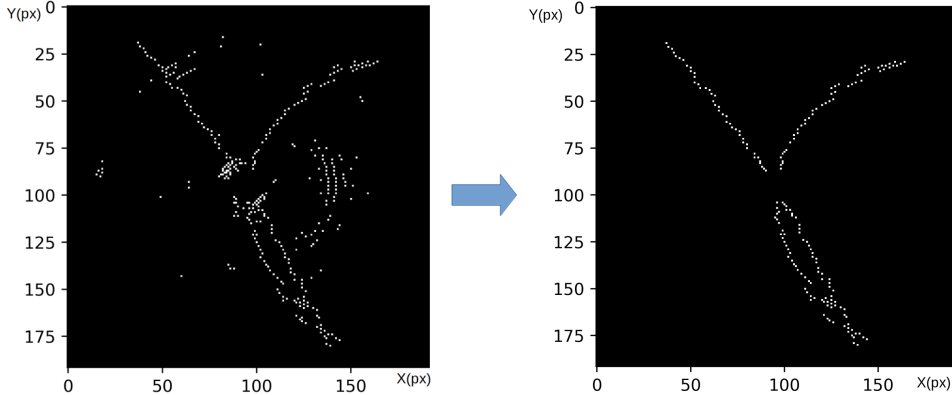


Figure 10: Left: a typical input image showing four tracks and noise hits. Right: desired output of the noise reduction algorithm demonstrating the succesful removal of noise hits.

resolution is big enough so that a single pixel does not need to store more than one measured hit in a wire. At the same time, 192 is a power of two, which is very convenient when applying 2x2 Max-Pooling as we do. Hence, an input image of the network will be a 192x192 image where all pixels storing a hit have value 1 and the rest have value 0.

Since we are dealing with supervised learning algorithms, each pixel in the input image must have a corresponding label, where the ground truth of the pixel is stored. It is only this way that the network can learn, by comparing its prediction for each pixel with the corresponding label. For each of the algorithms (noise reduction and track recognition) the labels will be different, since the desired outputs in each case are different. In figures 10 and 11 a visualization of the desired output for each algorithm is shown.

In the noise reduction algorithm, every pixel in the input image with an input value $x_{ij} = 1$ can have two different labels, denoted as y_{ij}: if a pixel is labeled as noise the label will have a value $y_{ij} = 0$, and if it is a track point the label will be $y_{ij} = 1$. However, in the track recognition algorithm we want the network to separate the different tracks in the image, so we need to label each track in a different way. Thus, in the track recognition algorithm the label for the noise point will still be $y_{ij} = 0$, but the track points will be labeled according to the particle track they belong to. For that, we assign a numeric value to each particle track, namely $y_{ij} = 1, 2, 3, 4$ for π^+, π^-, e^+, e^- respectively.

4.2 Output processing

Since we are using a sigmoid activation function in the output layer, the output of the network is an image with values between 0 and 1. This value corresponds to the 'confidence score' or probability of that specific point to be part of one of the particle tracks. The closer the value is to 0, the more likely is to be a noise point, and the closer to 1, the more likely to be a track point. In order to classify the points we need to apply a threshold, so that we only consider as track points pixels that have a value above that threshold. The threshold choice is quite important as it needs to be small enough to not misclassify too many noise points as track points but large enough so that we do not disregard too many track points. In the track recognition algorithm we have 4 output images, one for each particle track. Each of the images has pixels with values between 0 and 1 representing the confidence score of that pixel to be part of the track corresponding to that specific output image.

We do not care what will the network predict for the pixels that are not part of the original image, as we only want to classify these pixels as either noise or tracks. Thus, we remove from the output images all the points with value 0 in the input image leaving only the points representing a measured hit.

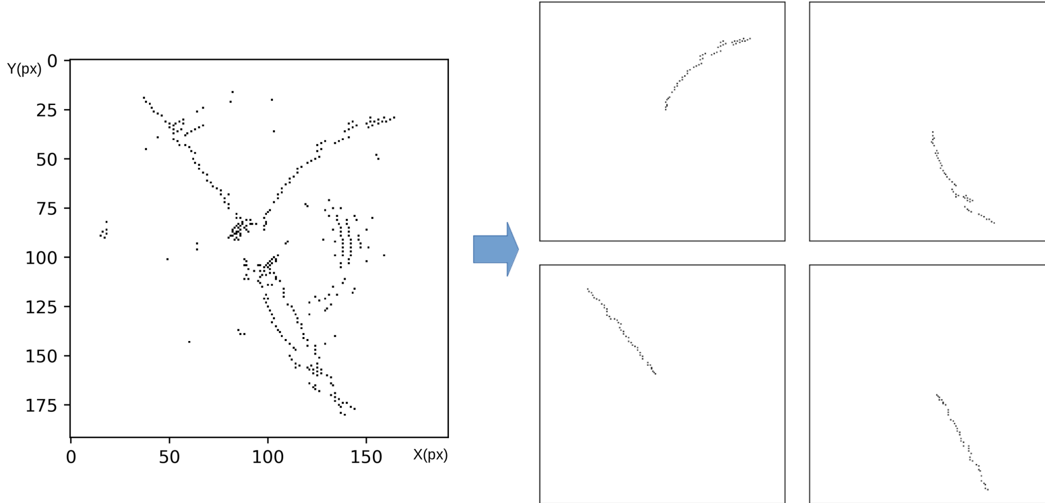


Figure 11: Left: typical input image showing four tracks and noise hits. Right: desired output of the track recognition algorithm demonstrating the successful identification of each of the particle tracks.

4.3 Loss functions

The loss function chosen for the noise reduction algorithm was the quadratic expression:

$$J(y,\hat{y}) = \frac{\sum_{ij,n}(\hat{y}_{ij} - x_{ij}y_{ij})^2}{\sum_{ij,n} x_{ij}} , \tag{10}$$

where $\hat{y}_{ij}$ is the pixel's true label, y_{ij} is the pixel's prediction and x_{ij} is the pixel's input value. The sums in i,j are taken over all the pixels in the input image and the sums in n are taken over all the images in the minibatch. By multiplying y_{ij} by x_{ij} we are not taking in account all the points not part of the original image, which have a value $x_{ij} = 0$.

For the track recognition algorithm a different loss function is used, directly based on the definition of the performance metric F1 score:

$$J(y,\hat{y}) = 1 - \frac{\sum_{ij,n} 2\hat{y}_{ij}y'_{ij}}{\sum_{ij,n}(2\hat{y}_{ij}y'_{ij} + \hat{y}_{ij}(1-y'_{ij}) + (1-\hat{y}_{ij})y'_{ij})} , \tag{11}$$

where $y'_{ij} = x_{ij}y_{ij}$ is the part of the predicted image that is also part of the original image.

4.4 Performance metrics

A neural network learns by minimizing the loss function, in this case the quadratic function shown in equation 10. We can consider this expression as a good indicator of how well the network is *learning*, but it is not a good indicator of how well the network *performs* overall. For instance, it does not take into account the chosen threshold. To properly measure the performance of the network we will use a different metric depending on which one of the two algorithms we are studying. All of the metrics we use are a function of the chosen threshold, and they can be maximized by searching for the optimal threshold in each case. We will always search for the optimal threshold through all the values in between 0 and 1 in steps of 0.01. This maximal value of the metric will be the value we assign to that metric.

The performance metric used in the noise reduction algorithm will be the so-called simple accuracy, defined as:

$$\text{accuracy} = \frac{\text{number of correctly predicted data points}}{\text{total number of points}} \tag{12}$$

For the track recognition algorithm the classification accuracy is not a good performance metric, since when there are 4 tracks each one corresponds to around 16% of the data points, and thus a 84% accuracy could be reached just by predicting that every point is not part of the track. A better way to measure the performance with a single value metric would be to use the definition of the F1 score:

$$\text{F1 score} = \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \cdot \text{ true positives}}{2 \cdot \text{true positive} + \text{false positive} + \text{false negative}}\ , \tag{13}$$

where the precision is the proportion of points predicted as track points that is actually correct and the recall is the proportion of points labeled as track points that is identified correctly, which can also be expressed as:

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}\ ; \qquad \text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negative}}\ , \tag{14}$$

The reason we use the F1 rather than both the precision and the recall is because it is usually more reliable to use a single-valued metric. If we were to use both precision and recall as metrics, there could be a case where one of them scores well and the other does not, making it difficult to evaluate performance. The F1 score, defined as an harmonic mean of the precision and the recall, is a good metric as long as both of them are equally important.

4.5 U-net architecture

In the master thesis [1], a search for an optimal neural network architecture for particle tracking in the BESIII main drift chamber is done. Two main fully convolutional network architectures were considered: a 'standard' CNN where the size of the images is conserved along the whole network, and a model inspired in the U-Net CNN, first proposed by Ronneberger et al [10]. U-Net consists of a contracting path where the size of the images is reduced and an expanding path where they are brought to their original size. We must note that it is neccesary for us that the output image keeps the same size as the input image. After comparing both architectures, U-Net was chosen as it outperformed the standar CNN. This was not surprising, as U-Net is the most prominent deep network in medical image segmentation and the most popular architecture in the field [17, 18], and it has been shown to perform very well even with a scarce amount of labeled training data [19].

There are two basic blocks of layers in U-net: convolutional blocks and up-sampling blocks. Each convolutional block consists of a convolutional layer, using a 5x5 convolution, 'same' padding and a s=1 stride, followed by a ReLu layer and a batch normalization layer. Up-sampling blocks consist of two layers: a first layer where 2x2 up-sampling operation is carried out, where every value in the image is repeated to a 2x2 grid, followed by a second layer with a 2x2 convolution also with 'same' padding and s=1 stride.

The number of filters in the convolutional blocks doubles after every max-pool layer and halves after every up-sampling block. This way, only the number of filters in the first convolutional block needs to be chosen. After a detailed hyperparameter search done in [1], a number of 24 filters in the first

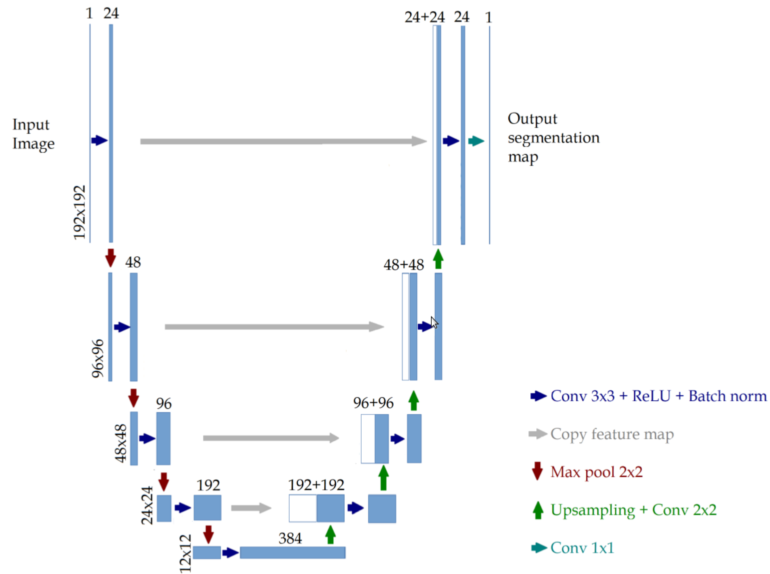


Figure 12: Adapted U-Net architecture for the noise reduction algorithm. For the track recognition algorithm, the only difference is that the last convolutional layer has 4 filters instead of 1, so that the network ouputs 4 feature maps, one for each track.

convolutional block and a 5x5 filter size for the convolutional layer in each convolutional block is chosen.

The overall architecture of the network (figure 12) is as follows:

1. A contracting path made up by four convolutional blocks that are each followed by a 2x2 max-pooling layer.
2. A expanding path consisting of four convolutional blocks that are each followed by up-sampling blocks.
3. One final convolutional block more followed by an output layer with one 1x1 filter.

In the layer in-between the contracting and the expanding path, images will have a small image size: the 192x192 input image is reduced to a 12x12 image with 192 channels. Therefore, a large number of filters can be applied without a huge computational cost.

5 Testing the model

In the master thesis [1], the network was tested with $\Psi(2S) \to 2\pi J/\Psi \to 2\mu 2\pi$ Monte Carlo simulated events from electron-positron anhilitations in BESIII. These were events with a low background noise level and only four particle tracks. In such conditions, the network was proven to perform with a very high efficiency. Specifically, when each algorithm was applied independently to the raw data, the noise reduction algorithm achieved a maximum accuracy of 97.5% while the track recognition algorithm scored a 96.0% F1 score when trained with 60.000 images. The performance was even higher when the noise reduction algorithm was applied first and the track recognition algorithm after.

Nevertheless, in the experiments which will be carried out in PANDA both the noise and the number of tracks per event will be much higher. In this section the algorithms will be tested on events

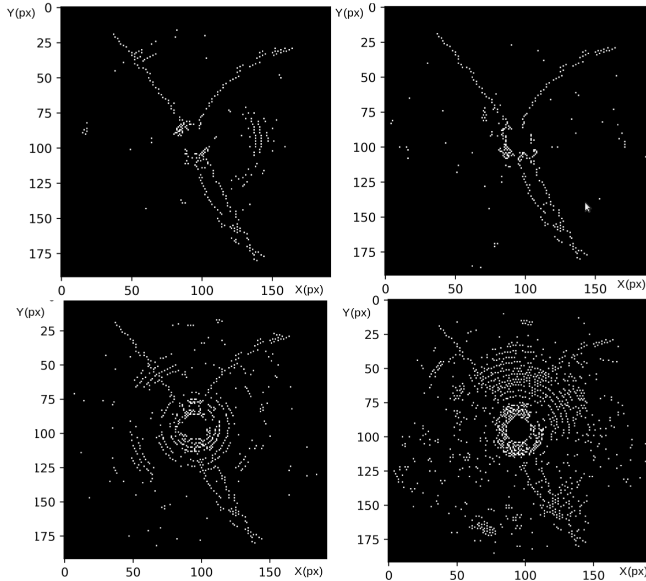


Figure 13: The same event with different noise ratios. From left to right and from the top to bottom, the noise ratios are 1:1, 1:2, 1:4 and 1:8.

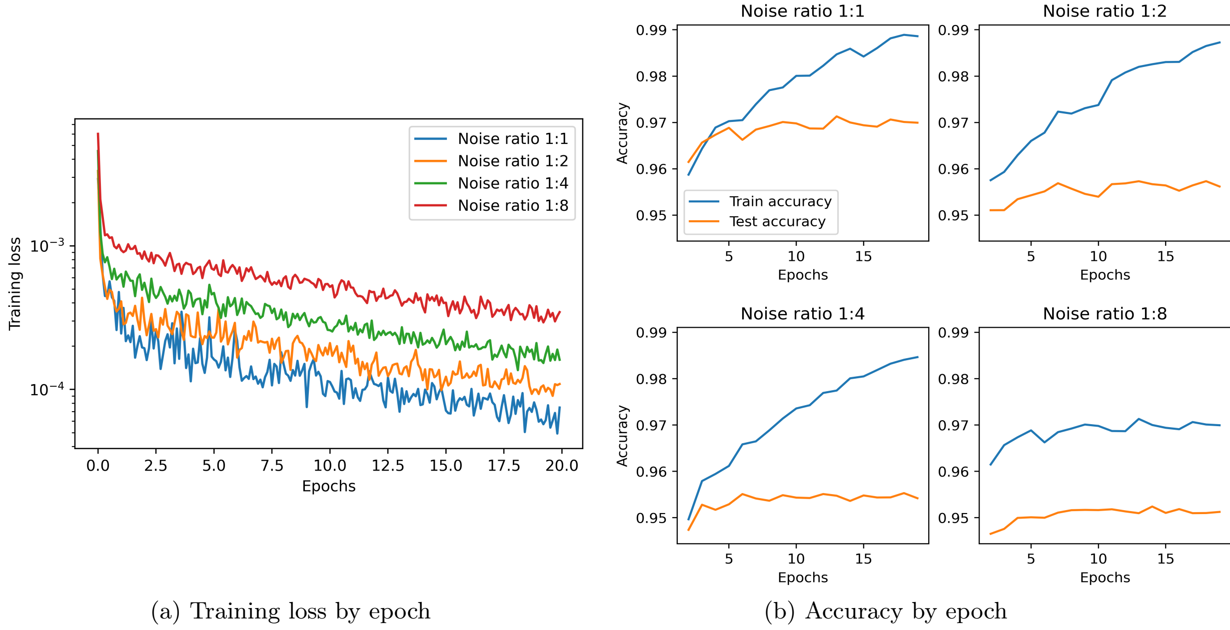
with variable conditions to try to establish the limits of its applications. In this project we used $\Psi(2S) \rightarrow J/\Psi\ \pi^{+}\pi^{-} \rightarrow e^{-}e^{+}\pi^{+}\pi^{-}$ events to train and test the CNN. The track points were generated through Monte Carlo simulations while the noise points are taken from real data that were triggered on random time signatures, not related to the electron-positron interaction point.

5.1 Testing on different background noise levels

To study the dependence of the performance of the network with the level of noise in the events, we generated 4 data sets with different background noise levels. These data sets all have the same track signal points but a different background noise level. We did so by taking each specific event and adding to it the noise points from other random events. Depending on how many 'noise events' we add to the events in each of the data sets we end up with data sets with different noise levels. Specifically we produced 4 different data sets with signal to noise event ratios 1:1, 1:2, 1:4 and 1:8 (see figure 13). This means that, for example, in the 1:4 data set the track points of each event are mixed with the noise points from 4 different random events. The 1:1 data have the same noise level as the data used in the master thesis. From now on we will refer to this feature as the 'noise ratio' of the data set.

Noise reduction

We trained the noise reduction algorithm in each of the 4 data sets for 20 epochs. The training was done in a training set of 10.000 images and a test set of 1.000 images, with both sets having the same noise ratio. The results of the training are shown in figure 14. The left panel compares the loss function measured in the training set (training loss) by epoch for each of the data sets. The CNN achieves a smaller training loss in the data sets with less noise after 20 epochs. In the right panel the accuracies by epoch of each data set are shown for both the test and the training set. The real performance of the network is measured by the test accuracy, as the test set is made up by data that the network has never seen before, just like the data generated in the experiments where this network could be used. The CNN scores a higher accuracy in the data sets with less noise, reaching a maximum test accuracy of 97.1% for the 1:1 noise ratio data set. For the data set with a 1:8 noise ratio, the maximum test accuracy is still



(a) Training loss by epoch

(b) Accuracy by epoch

Figure 14: Performance of the noise reduction algorithm in the 4 different data sets with signal to noise points ratios of 1:1, 1:2, 1:4 and 1:8.

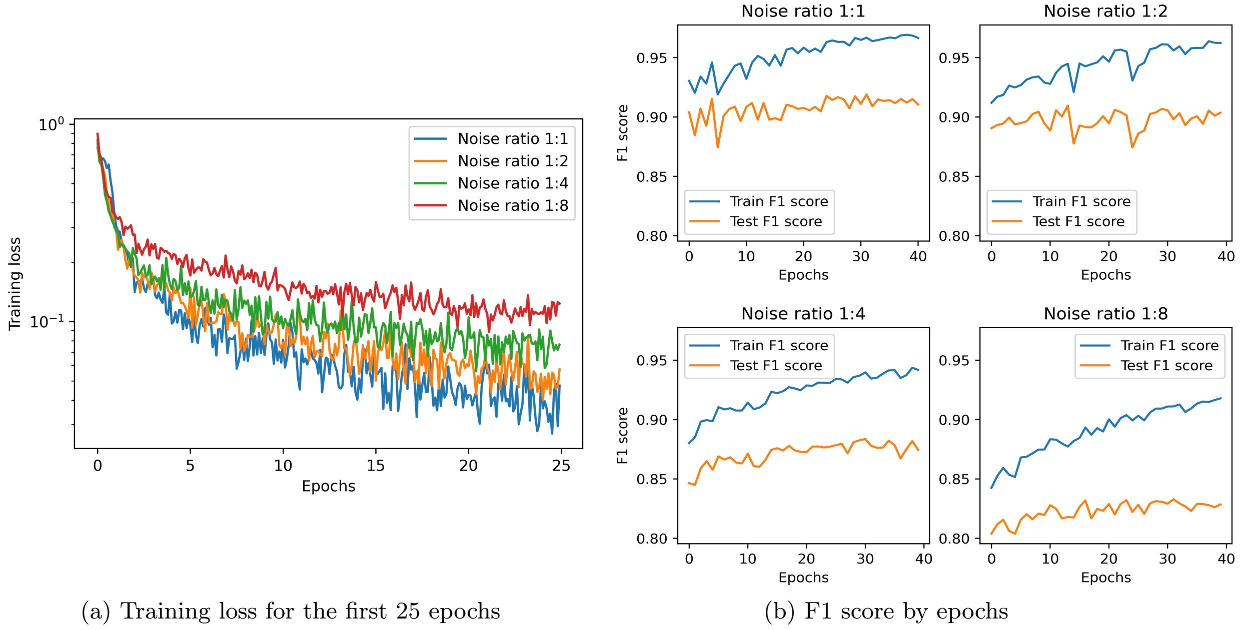
95.2%, demonstrating that the performance only drops slightly in the case the noise level increases. The maximum test accuracy for each data set is shown in table 1.

Track recognition

For the track recognition algorithm we trained the model for 40 epochs, as the learning process turned out to be slower than for the noise reduction algorithm. The results are shown in figure 15 and the maximum test F1 scores for each data set are summarized in table 1. In this case the test F1 scores decreases more significantly with increasing noise compared to the noise reduction algorithm. This demonstrates that the track recognition algorithm is more sensitive to the level of noise in the data.

	Noise ratio 1:1	Noise ratio 1:2	Noise ratio 1:4	Noise ratio 1:8
Noise reduction maximum test accuracy (%)	97.1	95.7	95.5	95.2
Track recognition maximum test F1 score (%)	91.5	91.0	87.2	83.4

Table 1: Maximum values of the noise reduction algorithm accuracy and the track recognition F1 score achieved in each of the datasets.



(a) Training loss for the first 25 epochs

(b) F1 score by epochs

Figure 15: Performance of the track recognition algorithm in the 4 different data sets with signal to noise ratios 1:1,1:2,1:4 and 1:8.

Noise reduction accuracy (%)		
	Applied to noise 1:1	Applied to noise 1:8
Trained with noise 1:1	97.1	94.2
Trained with noise 1:8	94.9	95.2

Track recognition F1 score (%)		
	Applied to noise 1:1	Applied to noise 1:8
Trained with noise 1:1	91.5	68.9
Trained with noise 1:8	84.2	83.4

Table 2: Performance of both algorithms when we train the network with events with a low noise/signal ratio 1:1 and then apply them to events with a higher noise/signal ratio 1:8, and viceversa.

Variable noise ratios

In PANDA, the generated events in experiments will likely have a variable noise. Thus, it is interesting to study if the network, when trained with events with low noise, can still perform well in other noisier events (and viceversa). With that goal in mind, we took the network trained in the previous section with the 1:1 data set and applied it to the 1:8 data set. Subsequently, we analyzed the opposite strategy by taking the network trained with the 1:8 data set and applying it to the 1:1 data set. This way we can measure how does the network perform when it is asked to filter noise and recognize tracks in data with a different level of noise than the data in which the network was trained with. The results for both algorithms are shown in table 2, where the corresponding performance metrics for each algorithm are shown in each case.

In both algorithms the network seems to perform better when the events used to train the network and the events in which the network is applied have the same noise ratio. When the network is trained with low noise events 1:1 (first row in the table), the performance is good when it is applied to the same noise ratio but it drops when applied to a higher noise ratio 1:8. This drop is particularly noticeable in the track recognition algorithm, where the F1 score goes from 91.5% to a 68.9%, decreasing more than 20%.

On the other hand, when the noise reduction algorithm is trained with high noise ratio 1:8 events, the performance in low noise events is only slightly worse than when trained with low noise (from 97.1% to 94.9%) while also performing good in high noise ratio. This suggests that for the noise reduction algorithm, training the network with noisy events helps the network to have a better overall performance along the different noise levels. Training the track recognition algorithm with noisy 1:8 events instead of with low noise 1:1 events yields a drop in the performance when applying to low noise events (from 91.5% to 84.2%) but greatly improves performance in noisy events (from 68.9% to 83.4%).

In conclusion, these results suggest that if a fixed noise level is expected for the events in the experiment, then the network should be trained with events with a similar noise level, since we have seen that this yields the best results. On the other hand, if the events with variable noise are expected, then training with noisy events is the most efficient method to have a good overall performance.

5.2 Adding random tracks

In PANDA, the data rates will be so high that events are likely to overlap, resulting in images with many tracks. Consequently, it is interesting to know if our network is able to filter tracks that are not part of the studied event. To do so, we produced a new data set by taking the previous 1:1 noise ratio data set and adding to each event a new random track from another event. This new random track is taken from another random event and can correspond to any of the 4 particles e^-, e^+, π^+, π^- (figure 16). We would like to see if the network is able to recognize this fake track as noise to remove it from the image.

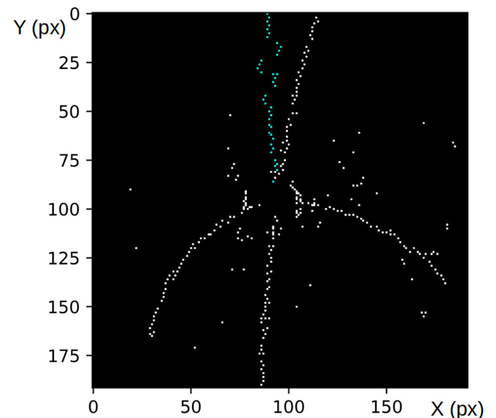
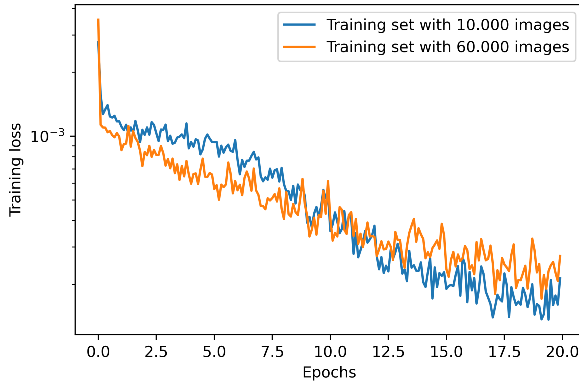
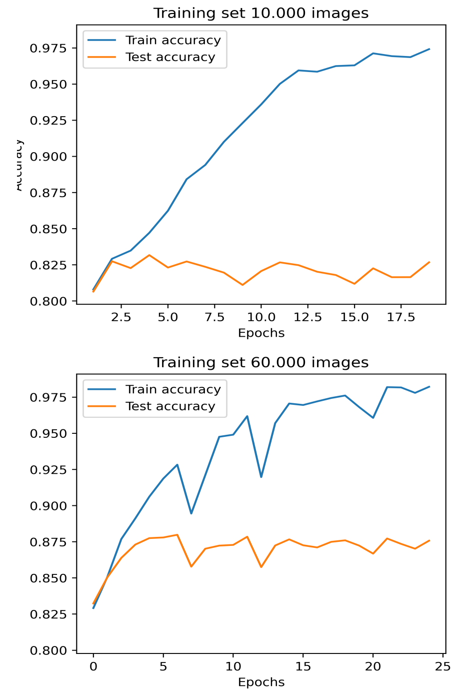


Figure 16: Visualization of a 1:1 noise ratio event with a fake track added (in cyan). In this case the added track corresponds to a π^+ particle.

We want the network to classify the new random track as noise. Thus, both the background noise and the random track should be labeled equally. We first trained the model with a training data set of 10.000 images and a test set of 1.000 images for 20 epochs. For the noise reduction algorithm, the network achieves a test accuracy with a maximum of 83.2%, and for the track recognition algorithm a F1 score with a maximum of 80.8%. Both cases yield worse results than when no fake track is added. It is clear that it difficult for the model to filter a proper track randomly added to the event than the background noise. The network seems to learn well the training set, as both the train accuracy for the noise reduction and the train F1 score for the track recognition algorithm are high, but it is clearly not able to generalize to the test set; the network is overfitting the data. Therefore, as a next step we trained the CNN with a larger training set, as this is usually a good strategy to avoid overfitting.

We trained again the model with the same data set but now with a training set of 60.000 images for the noise reduction algorithm and with 25.000 images for the track recognition (it was not possible to train the latter with more images due to memory limitations). We reached a maximum accuracy of 87.0%for the noise reduction algorithm and a 84.1% F1 score in the track recognition algorithm. The comparison between the training with 10.000 and with 60.000 images is shown in figure 17 for the noise reduction algorithm and in figure 18 for the track recognition. By using a larger training set the performance was improved. However, after visualization (see figure 19), we observe that the algorithm primarily filters the random noise hit but not the added track.

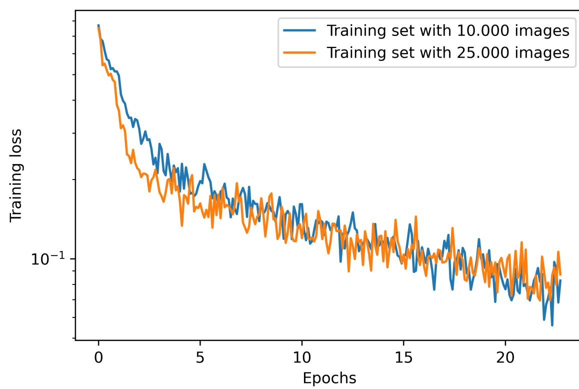




(a) Training loss by epoch

(b) Accuracy by epoch

Figure 17: Comparison of the noise reduction algorithm when trained with 10.000 and 60.000 events with an added track and a noise ratio 1:1. Test set in both cases comprises 1.000 images.



(a) Training loss by epoch

(b) F1 score by epoch in both the training with 10.000 and 60.000 images

Figure 18: Comparison of the track recognition algorithm when trained with 10.000 and 25.000 images with an added track and a noise ratio 1:1. Test set in both cases comprises cases comprises 1.000. images

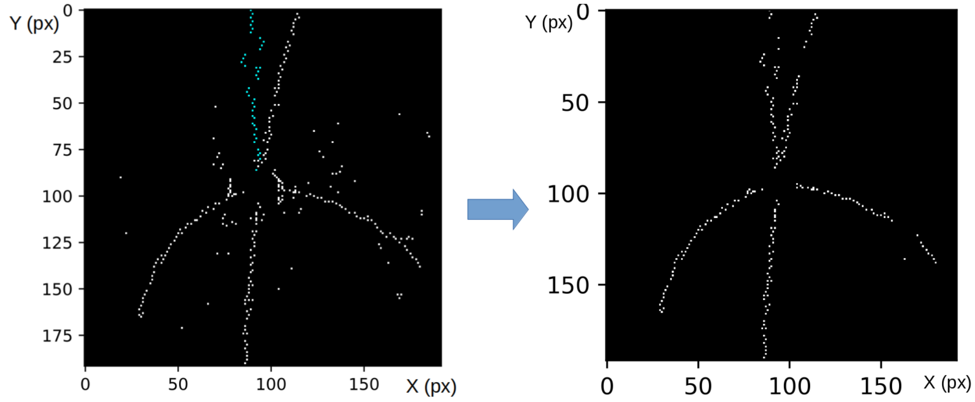


Figure 19: Left: input of the network showing the added track in blue. Right: the output of the network demonstrating how the CNN is not able to filter the added track, while still filtering the noise.

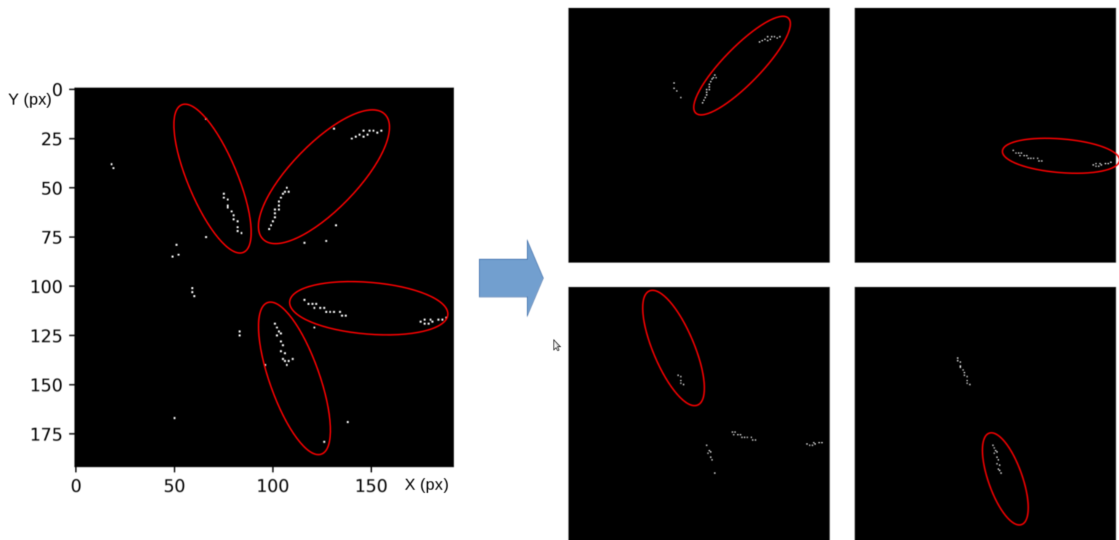


Figure 20: Left: input image of the network consisting of an incomplete event where the hits measured with the tilted wires have been erased. Right: output images of the CNN where the tracks are not properly identified, demonstrating the incapability of the network to recognize the different tracks in incomplete events. The different particle tracks are marked inside red ellipses.

It is not surprising that the network has more difficulties to filter these fake added random tracks than in the case of the background noise. The patterns of these tracks have very similar features than the ones of the event itself. In order to distinguish them, the CNN has to be able to measure the momentum of the different particles to see which one does not add up to the initial momentum. This means the network has to learn about momentum conservation from the images. Extracting these high-level features, related with the relative orientation of the different tracks, is not easy to learn for the CNN. On the other hand, filtering the background noise involves learning low-level, local features related to connectivity between the points, for which the CNN works nicely.

5.3 Testing on incomplete events

In this final section we studied the performance of the track recognition algorithm in incomplete events. The goal is to see if the algorithm is still able to identify the different particle tracks in events where some

sections of the tracks are missing. We will assume that the tilted wires of the MDC are not working, so the input images of the network will only be composed of the hits measured with the axial wires.

To train the CNN we used 10.000 complete events with a 1:1 noise ratio. Applying the algorithm to a test set of 5.000 incomplete events yielded a F1 score of 45.9%. A visualization of one of the predicted images by the CNN can be seen in figure 20. The algorithm tested in the same data set but with the events showing all the points from the tilted and the axial wires scored a F1 score of 92.5%. This shows that the algorithm is in general not able to identify individual tracks in events where the hits measured by the tilted wires are not available. This demonstrates again that the network works nice with local features, but it breaks down when it is asked to extrapolate more global features, such as connecting different sections of the incomplete tracks.

6 Conclusions

In this research project the application limits of a convolutional neural network for the analysis of tracking data in detectors such as PANDA or BESIII were studied. The CNN was tested with Monte Carlo simulated data from the decay channel $\Psi(2S) \to J/\Psi\ \pi^+\pi^- \to e^-e^+\pi^+\pi^-$ in the BESIII experiment, and the noise was taken from real data in the same detector. We have studied the applications of the CNN using two algorithms: noise reduction and track recognition. These algorithms were already studied in [1], achieving good performance scores in relatively clean events. In this project we tested their performance in events with more extreme conditions such as those expected in modern experiments like PANDA or those in CERN, where the high event rates causes overlapping between events.

Firstly, we studied the dependence of the network's performance on the background noise levels of the events. The noise reduction algorithm showed great accuracy even when it was applied to very noisy events (over 95% of accuracy), while the performance of the track recognition algorithm did not scale as good when the noise level for each event was increased. We also demonstrated the importance of the noise level of the events used to train the CNN, as using low noise events in the training set was shown to decrease the performance when the CNN was applied to noisier events. Then, the number of tracks in the events was increased from 4 to 5 by adding a fake track, and the network was trained to filter this added track. To do so, the network should be able to extract high-level features from the images, such as learning the momentum conservation law, to distinguish the fake track from the real tracks. The network was unable to filter this added track efficiently. Finally, we applied the track recognition algorithm to incomplete events were the hits measured by the tilted wires had been erased. In order to be able to identify the tracks, the network should extrapolate the missing points of the tracks, using information from all the tracks. The CNN was not able to recognize the incomplete tracks, demonstrating again its incapability to work with high-level, global features built with information from all the tracks.

In conclusion, the CNN studied in this project was shown to be a potential candidate as a noise reduction algorithm given its good performance filtering background noise even in very noisy events. It could also be used as track recognition algorithm to segment the different particle tracks in events with low noise, or in events where the background noise has been reduced by the noise reduction algorithm. Nevertheless, since the CNN is not able to filter added tracks not part of the event, it could not be used in events such as those expected in PANDA or other modern experiments, where the events will likely overlap generating images with many particle tracks. Nonetheless, this doesn't mean the CNN methodology cannot be applied on such events, since this network could be improved with a more careful tuning of its hyperparameters or by taking in consideration other CNN architectures apart from U-Net.

References

[1] H de Vries. *Convolutional Neural Network for Reducing Noise and Detecting Tracks in the BES-III Main Drift Chamber*. 2019.

[2] PANDA collaboration. *Physics Performance Report for PANDA: Strong Interaction Studies with Antiprotons*. 2009

[3] A Herten. *GPU-based online tracking for the PANDA Experiment*. 2015.

[4] K Albertsson et al. *Machine Learning in High Energy Physics Community White Paper*. 2018.

[5] P Dalpiaz. *Charmonium and other onia at minimum energy*. Physics With Cooled Low Energetic Antiprotons, edited by H. Poth, pages 111–124. 1979.

[6] PANDA collaboration. *Technical Design Report for the PANDA Forward Tracker*. 2017.

[7] BESIII collaboration. *Physics at BES-III*. 2008.

[8] C Chen et al. *The BES-III drift chamber*. 2007 Nuclear Science Symposium Conference Record, 3:1844–1846. 2007.

[9] K O'Shea, R Nash. *An Introduction to Convolutional Neural Networks*. 2015.

[10] O Ronneberger, P Fischer, T Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015

[11] R Krishna, D Shu, F Li. Notes on the course *CS231n: Convolutional Neural Networks for Visual Recognition*. Standford University, Spring 2020.

[12] Y LeCun, Y Bengio, G Hinton. *Deep learning*. Nature. 2015.

[13] X Glorot, A Bordes, Y Bengio. *Deep sparse rectifier neural networks*. 2011.

[14] S Shi, Q Wang, P Xu, X Chu. *Benchmarking state-of-the-art deep learning software tools*. 2016.

[15] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016.

[16] N Qian. *On the momentum term in gradient descent learning algorithms*. 1999.

[17] M Drozdzal, E Vorontsov, G Chartrand, S Kadoury, C Pal. *The importance of skip connections in biomedical image segmentation in Deep Learning and Data Labeling for Medical Applications*. 2016

[18] N Ibtehaz, M S Rahman. *MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation*. 2019

[19] G Litjens, T Kooi, B E Bejnordi, A Setio, F Ciompi, M Ghafoorian, J A Van Der Laak, E Van Ginneken, Clara I Sánchez. *A survey on deep learning in medical image analysis*. Medical image analysis. 2017.

[20] He et al. *Deep residual networks for image recognition*. 2015

[21] S Ioffe, C Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015

[22] S Santurkar, D Tsipras, A Ilyas and A Madry. *How Does Batch Normalization Help Optimization?*. 2018

[23] I Goodfellow, Y Bengio, A Courville *Deep Learning*, section 8.7.1. MIT Press. 2016. http://www.deeplearningbook.org

[24] D P Diederik, J Ba. *ADAM: a method for stochastic optimization*. 2016.

[25] A Krizhevsky, I Sutskever, G Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. Advances in Neural Information Processing Systems 25 1090–1098. 2012.