**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**
**WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI**
INSTYTUT INFORMATYKI

# Praca dyplomowa

*Application of machine learning techniques to improve the resolution of particle timing detectors*

*Zastosowanie algorytmów uczenia maszynowego w celu poprawy rozdzielczości detektorów czasu przelotu cząstek*



Autor: Mateusz Kocot
Kierunek studiów: Informatyka
Opiekun pracy: dr Leszek Grzanka

Kraków, 2023

# Acknowledgements

I would like to express my deepest gratitude to Leszek Grzanka, the supervisor of this thesis, for providing me with the opportunity to collaborate with CERN and for his assistance in solving all project-related problems, not only the technical ones.

I am grateful to Valentina Avati, Edoardo Bossini, and Nicola Minafra for their clear explanations, brilliant ideas, and the friendly atmosphere they created during our calls and meetings.

The project was partially funded by the Polish Ministry of Education and Science, project 2022/WK/14.

# Abstract

This thesis investigates the use of machine learning techniques to improve the precision of timing measurements in the Precision Proton Spectrometer (PPS) at CERN's Large Hadron Collider (LHC). PPS is a subsystem of the Compact Muon Solenoid (CMS) detector and is responsible for detecting so-called forward protons. PPS uses diamond-made timing sensors and advanced readout electronics to produce sampled signals of voltage when detecting a particle. These signals are measured with varying amplitudes, which introduces the effect called time walk. The current method of mitigating the time walk effect – the constant fraction discriminator (CFD) – has certain drawbacks, such as not taking into account noise, or being unable to handle signal properties that depend on amplitude. Machine learning approaches have the potential to improve the handling of the time walk effect and mitigate the CFD's issues.

The practical part of this thesis involves predicting a single timestamp based on 24 values of sampled voltage. Deep neural networks are the focus of the experiments, as they have been shown to be effective in similar applications. Various deep learning architectures were tested using a hyperparameter tuning procedure incorporating the Bayesian optimisation strategy and cross-validation. Ultimately, models based on the UNet convolutional architecture yielded the best results.

The study provides a comprehensive analysis of the optimal networks outputted by the tuning procedure. The networks were compared to the current state-of-the-art – CFD – in a series of numerical experiments and consistently demonstrated improved timing precision, despite being trained on a low-quality reference dataset. The improvements ranged from 6% to 18% in the primary numerical experiment, providing strong evidence that neural networks can be effectively used in the timing measurements at PPS.

# Contents

# 1. Introduction

The purpose of this chapter is to state the research goal and introduce the reader to the basic concepts required to understand the thesis.

## 1.1. CERN

The French acronym CERN was born in 1952 [1] when the European Council for Nuclear Research was established. Even though the Council transformed into the European Organisation for Nuclear Research in 1954, the acronym was retained. The CERN laboratory – the largest particle physics facility in the world – is based close to Geneva, on the France-Switzerland border. CERN paved the way for multiple discoveries, including the observation of the Higgs boson in 2012 [2, 3] or the discovery of odderon in 2020 [4]. CERN is also known for being the place where the World Wide Web (WWW) was invented in 1989 followed by launching the first website in 1991.

The CERN laboratory operates numerous particle accelerators, including the famous Large Hadron Collider (LHC) [5], which is the most powerful accelerator on Earth. It is located in a 27-kilometre, ring-shaped tunnel, at an average depth of 100 m. Inside the tunnel, two particle beams travel in opposite directions in two separate beam pipes. Superconducting magnets and various accelerating structures guide the particles around the ring and boost their energy. The LHC pushes hadrons (protons or ions) to near the speed of light.

The LHC life cycle is divided into operation periods called Runs and Long Shutdowns between them. During the shutdown periods, the capabilities of the LHC and its detectors are improved, so that the following Run can bring even more spectacular measurements to life. Since its relaunch into Run 3 in 2022, the LHC is capable of accelerating protons up to 6.8 TeV which stands for around 99.999999% of the speed of light. This allowed measuring collisions with the unprecedented energy of 13.6 TeV in July 2022 [6].

Accelerated particles can collide at four points where the LHC beam pipes intersect. These points are called Interaction Points (IPs). Every collision produces a myriad of new particles. Huge particle detectors were built around the IPs to capture all these particles and produce petabytes of fascinating data. One of these central detectors is the Compact Muon Solenoid (CMS) [7]. Together with another major detector – ATLAS [8] – CMS played a crucial role in observing the Higgs boson.

## 1.2. Precision Proton Spectrometer

Central detectors like CMS are perfectly suited for the exploration of events in which the protons collide centrally producing a swarm of new particles. However, in a sizeable fraction of these proton-proton events called CEP (Central Exclusive Process) events, the collisions have only a small impact on the protons' trajectories, hence they remain almost intact and continue their movement together with the beam. Such protons are called 'forward' or 'scattered', because they scatter to minimal angles, lose only a tiny bit of their energy and continue almost forward. These particles cannot be detected at the IPs, as they travel together with the beam. However, since they lost a fraction of their energy, they divert from the beam as the distance from the IP increases. Thus, in

order to detect them, detectors have to be placed a few hundred metres from an IP which allows the forward protons to gain a distance of a few millimetres from the beam.

The Precision Proton Spectrometer (CMS-PPS or just PPS) [9] is a subsystem of CMS [10] capable of detecting the forward protons. It used to be a joint project of CMS and TOTEM (TOTal Elastic and diffractive cross section Measurement) [11], so it is often referred to as CTPPS (CMS-TOTEM-PPS), too. The sensors of PPS are hosted in retractable vessels called Roman Pots (RPs) [12]. RPs are mounted at a distance of between 210 and 220 m from the CMS Interaction Point. They are inserted into the beam pipes close enough to capture forward protons, but, on the other hand, far enough not to touch the centre of the beam which would cause destructive radiation damage to the sensors (see Figure 1.1). The RPs are placed symmetrically on both sides of the CMS IP, in the LHC sectors[1] 45 (anticlockwise of the CMS IP) and 56 (clockwise of the CMS IP).

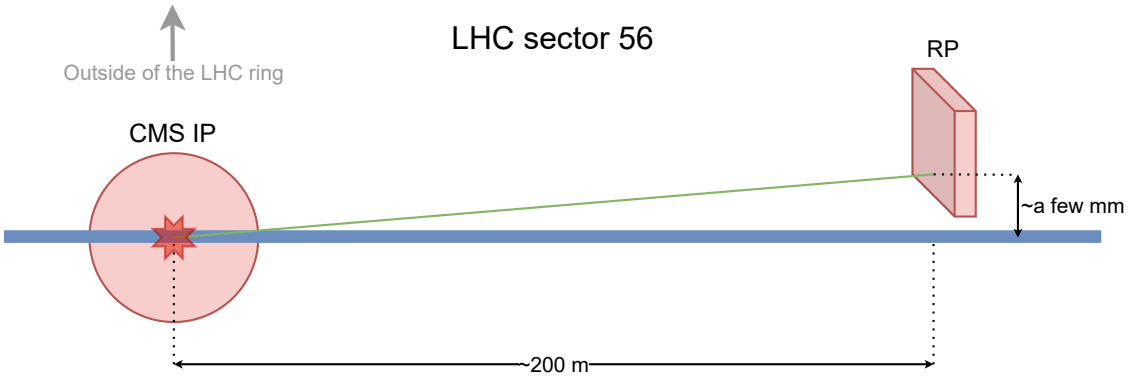


**Figure 1.1:** Simplified diagram showing the detection of a scattered proton (green trace) by a PPS RP inserted safely at a distance of a few mm from the LHC beam (blue). Only one side of the CMS IP is shown (sector 56), but a symmetric layout could be also drawn on the other side (sector 45).

There are two types of RPs used in PPS. The tracking RPs are used to measure the exact positions of forward protons with respect to the LHC beam. On the other hand, the timing RPs provide precise time measurements. A timing RP is responsible for measuring the time of arrival, which stands for a precise timestamp of a particle flying through the detector. This value can be used to reconstruct the time of flight, which is the time difference between the collision and capturing the particle in the detector.

## 1.3. Timing measurements in PPS

Each timing RP of PPS is equipped with four parallel detector planes [13] (see Figure 1.2, left) inserted perpendicularly to the beam. Thus, a particle can be detected even four times within an RP which improves the time precision of a particle measurement. Every plane hosts 12 timing sensors in the shape of a strip, placed at increasing distances from the beam (see Figure 1.2, right top). The strip width is associated with the beam intensity at a particular distance. The further from the beam, the fewer the number of flying particles. Therefore, the far strips are broader than the closer ones.

[1] The whole LHC ring is divided into 8 sectors.

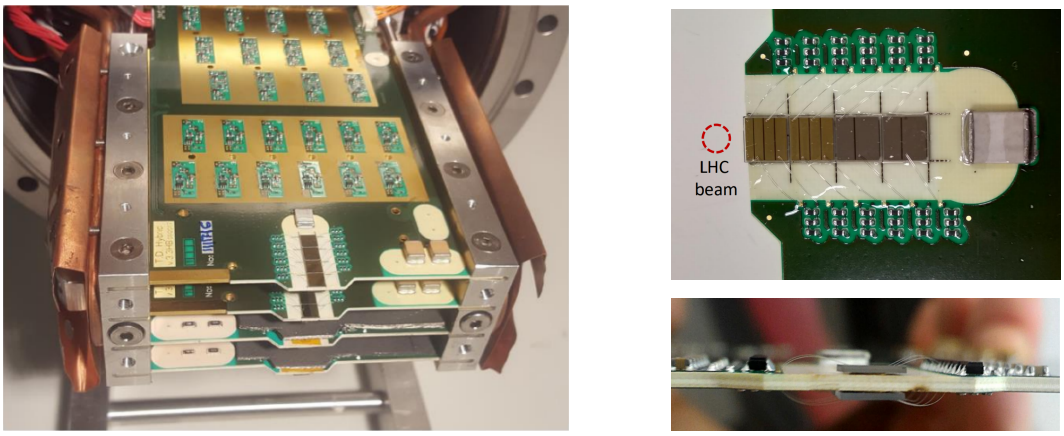


**Figure 1.2:** Left: complete set of four detector planes with timing sensors hosted in an RP; right top: strip sensor layout on a single plane; right bottom: diamond sensors mounted on both sides of a plane and wires connecting them to the readout channels. The same channels are used for the respective diamond sensors on both sides, hence the name: 'double-diamond'. Source: [14] (left), [13] (right top), [15] (right down)

The strip sensors are made of high-purity synthetic diamonds [16]. Due to the semiconductor properties of diamond, when a particle passes through a properly designed diamond sensor, it induces a current on the sensor's electrodes. The signal generated by a passing particle is then amplified and digitised. Each sensor is connected to a single readout channel resulting in 12 channels per plane and 48 channels in an RP. As from the LHC Run 3, every strip is a so-called double-diamond (DD) sensor. This means that diamond sensors are mounted on both sides of the detector plane but connected to the same amplification channel (see Figure 1.2, right bottom). Such a sensor architecture doubles the strength of a signal produced by a flying particle while keeping the noise on an unchanged level.

All these channels are connected to a discriminator coupled to a time-to-digital converter (TDC). The goal of a discriminator is to trigger a TDC when a particle signal is detected. The TDC is designed to provide the digitised timestamp of the signal. PPS selected the NINO [17] chip for the discrimination phase and the HPTDC [18] for the second step. The HPTDC is capable of detecting both the leading and trailing edges of the signal. This method is simple and efficient, but it outputs only two samples from the whole signal. A different approach is based on sampling the signal more frequently using the SAMpler for PICosecond time pick-off (SAMPIC) chip [19] capable of producing the whole time series of signal voltage. In the case of the PPS setup at the LHC, SAMPIC produces time series of 24 voltage samples spanning across the 3 ns period. Unfortunately, due to the data throughput limitations, only two channels from each plane are connected to SAMPIC. The readout layout of the PPS timing RPs is presented in Figure 1.3.

## 1.4. Time walk effect

The goal of the system described above is to output the time of arrival of a particle. The easiest approach would be just to use the leading signal edge timestamp digitised by HPTDC. However, this method, referred to as a fixed threshold algorithm, highly depends on the signal amplitude. The bigger the amplitude, the shorter it takes for the signal to reach a selected threshold. This effect

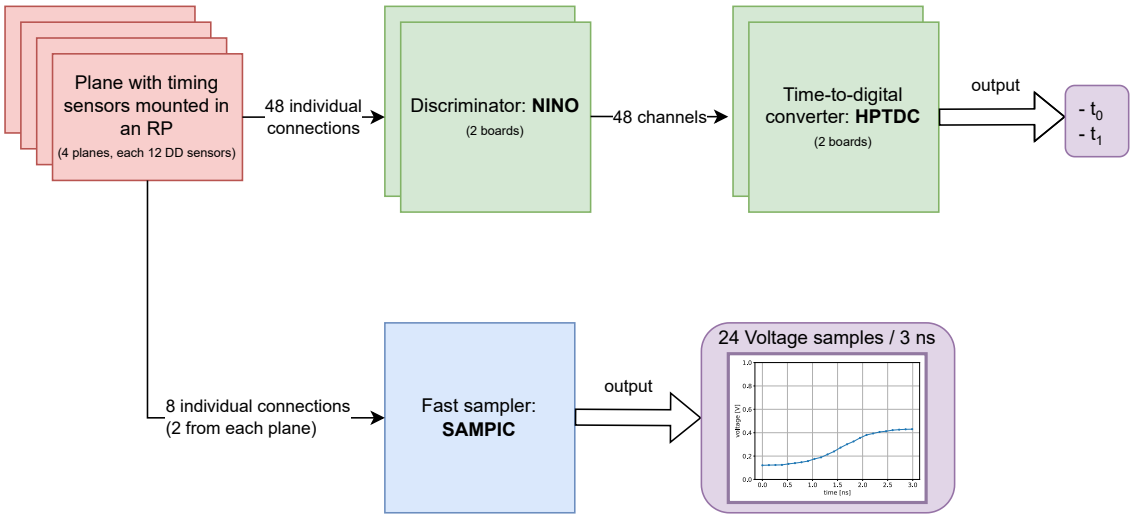


**Figure 1.3:** Readout layout of a PPS timing RP. HPTDC outputs two numbers: signal leading edge timestamp ($t_0$) and trailing edge timestamp ($t_1$). The final output from SAMPIC is a time series of sampled Voltage.

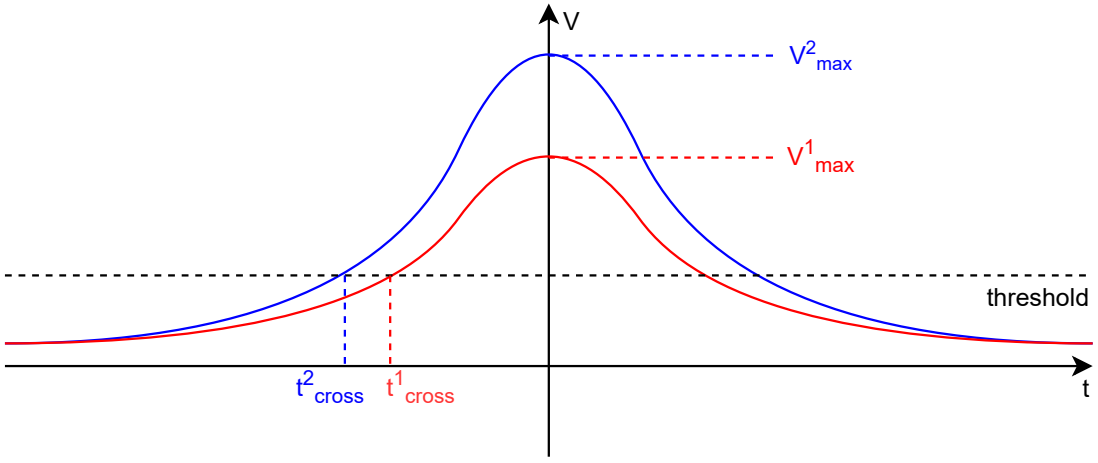


**Figure 1.4:** Example of the time walk effect on two signals with the same shapes, but different amplitudes. Even though the signal 1 (red) and 2 (blue) reach their maximums at the same time, due to their difference in amplitude ($V^1_{\max} < V^2_{\max}$), they cross the given threshold at different times ($t^1_{\text{cross}} > t^2_{\text{cross}}$).

is known as the time walk effect (see Figure 1.4). A particle passing through four detector planes can induce signals with different amplitudes which would make the results of the fixed threshold algorithm inconsistent.

PPS uses two methods to minimise the impact of the time walk effect. In the case of HPTDC readout, the time-over-threshold (TOT) method is used. The time-over-threshold value can be easily computed by subtracting the rising edge timestamp from the trailing edge one. TOT uses a special formula of the leading edge timestamp and the time-over-threshold value to handle the time walk effect. The biggest drawback of this method is taking only two signal samples into account. Nonetheless, it utilises all the data available in the HPTDC output. While TOT could also be used for the SAMPIC outputs, there are better methods using more samples from the time series.

The algorithm used presently in PPS is the constant fraction discriminator (CFD) implemented as the Normalised Threshold algorithm with several improvements. In short, it reduces the time walk effect by normalising signal amplitudes to one value and then following the fixed threshold approach.

## 1.5. Thesis goal

CFD is successfully used in PPS. Nevertheless, it has certain drawbacks. First of all, the implementation used currently at PPS does not have any specific mechanism to reduce the effects of signal noise except for using the average of the first samples to bring the signal baseline to 0. Other than that, noise is not considered when finding the amplitude or finding the timestamp of crossing the threshold. Sophisticated interpolation methods could be used to account for noise. However, they require expert knowledge and cannot be used continuously for all the measurements. Small changes in the detector setup or the ageing of its components could render all the previous interpolation efforts outdated. Another weakness lies in the biggest strength of CFD – normalisation. Some signal properties might depend on the signal amplitude, but CFD is not capable of taking them into account. Finally, only the samples close to the amplitude and the threshold-crossing are used, and the shape of the signal is not considered. Similarly to the amplitude, the signal shape might be relevant when computing the particle timestamp.

All of these drawbacks could be mitigated by using machine learning (ML) techniques. Due to their ability to adapt to existing data, they are capable of reducing the time walk effect and taking into account even the most subtle signal properties. **The goal of this thesis is to investigate the possible ML approaches and compare them with the current state-of-the-art – CFD.** The practical part of the problem lies in the analysis of the time series and predicting a single timestamp based on 24 values of sampled voltage outputted by SAMPIC.

# 2. Deep neural networks

Following the initial literature study, the machine learning domain is limited to deep learning, which means that the experiments include various architectures of deep neural networks. This chapter briefly introduces deep learning concepts required to comprehend the following chapters of the thesis.

## 2.1. Perceptron

Perceptron [20] is the simplest neural network (NN) type used to model a mathematical function. It was designed as a mathematical model of a biological neuron. It takes a vector $\mathbf{x}$ as input and outputs either $0$ (false) or $1$ (true), so it can work as a classification algorithm, or in short – classifier. Inside, a perceptron uses a vector of weights $\mathbf{w}$ and a bias $b$ to calculate the final value $f(\mathbf{x})$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{w} \cdot \mathbf{x}$ is a dot product of the two vectors. In essence, a perceptron is just a linear regression algorithm with the step function applied to the result, although, in principle, the step function does not have to be applied. Such a simple algorithm can already be called a neural network with one neuron since there is a single number on the output (see Figure 2.1). Each weight from the vector $\mathbf{w}$ decides the importance of the corresponding variable from the input vector $\mathbf{x}$. Unfortunately, due to its simplicity, a perceptron can model only functions that are linearly separable.

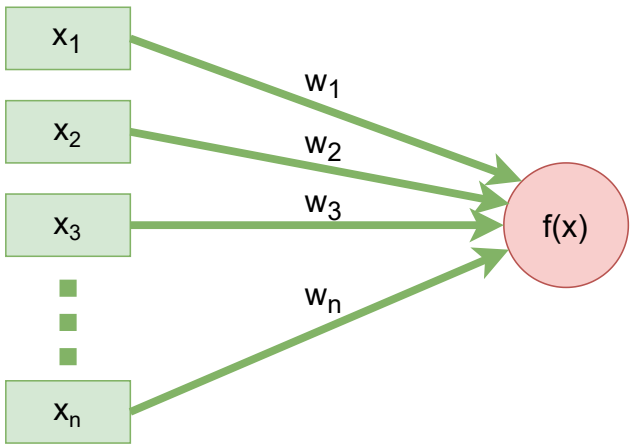


**Figure 2.1:** Perceptron scheme. $f(x)$ is a weighted $(w_1, ..., w_n)$ sum of inputs $(x_1, ..., x_n)$ plus a bias which is not shown in the picture.

## 2.2. Multilayer perceptron

To overcome the issues of a single perceptron, many perceptrons can be stacked together building a multilayer perceptron (MLP). Every layer of an MLP is a group of perceptrons (neurons), each with a separate vector of weights and a bias. All the weights from the layer can be collected in a

matrix $\mathbf{W}$, and the biases in a vector $\mathbf{b}$. The formula for an MLP layer with $n$ input variables and $m$ separate perceptrons, or, in other words, $m$ output neurons is:

$$f(\mathbf{x})_{m\times 1} = \mathbf{W}_{m\times n} \cdot \mathbf{x}_{n\times 1} + \mathbf{b}_{m\times 1}$$

Each output neuron is connected with every input variable. Therefore, such a layer is called fully connected, densely connected or just dense.

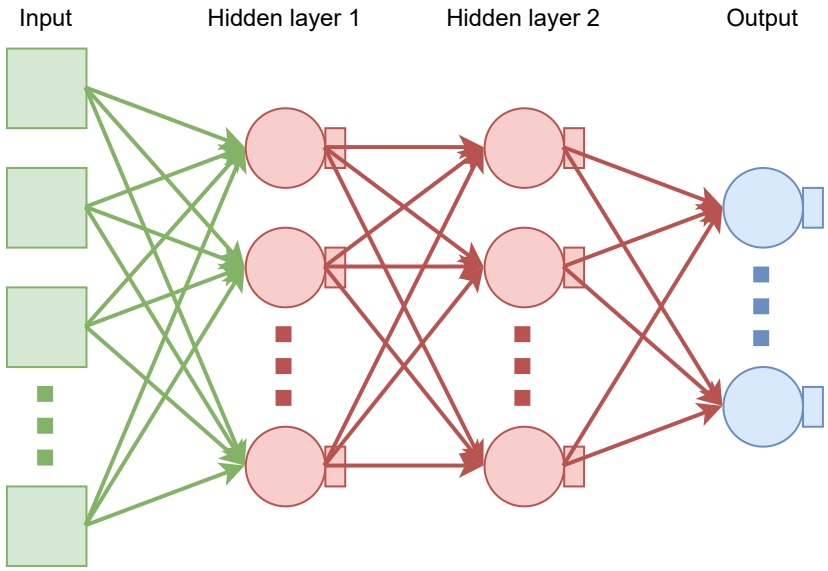


**Figure 2.2:** Example of a multilayer perception with two hidden layers. Activation functions are marked with vertical rectangles (red: internal activation, blue: output activation).

An MLP makes use of several layers stacked together so that the output of one layer is the input for the next one (see Figure 2.2). The first layer is the input, the last one – the output layer, and the middle ones are called 'hidden'. The layers cannot be just stacked without any operation between them, since a compound of linear functions is still a linear function. No matter how many layers would be used, they could still be represented as a linear function. Therefore, a so-called activation function is used after each layer to introduce nonlinearity. It has been proven that such an MLP is a universal approximator, which means that, if constructed properly, it can model any mathematical function [21].

In principle, any nonlinear function could be the activation function. However, for the reasons described in the following paragraphs, the function has to be differentiable. Today, the most common activation function is ReLU [22] (see Figure 2.3, left), however sigmoid (see Figure 2.3, right) used to be employed frequently in the past and still has numerous use cases nowadays. It is not necessary to add an activation function after the last layer in an MLP. In regression problems, the output from the whole network is usually just a single number. However, sometimes it is required to perform some operations on the output values. For instance, in binary classification, it is worth applying the sigmoid after the last layer so that the final value is a number from the $(0, 1)$ range. This number can stand for the probability of class $1$ (true).

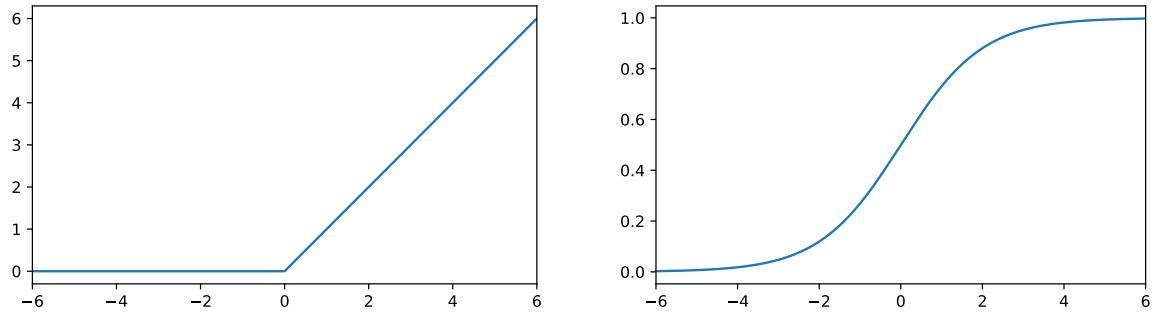


**Figure 2.3:** ReLU (left) and sigmoid (right)

## 2.3. Training a neural network

As mentioned above, an activation function has to be differentiable. The reason for that is the way an MLP is trained. Training a neural network means adjusting the connection weights so that the output of the whole network is satisfying for a given input. For instance, for a dataset containing images of cats and dogs, the expectation might be to predict the correct class (assuming one class is 0, and the other – 1). The training goal is to fit the network weights to available data so that it can also generalise – work for new, unseen data. Usually, the means of verifying if the network is capable of reasonable predictions for new data is computing its accuracy, which stands for the number of examples for which the prediction was correct (e.g. predicting the correct animal) divided by the number of all the examples in the dataset. In the case of a regression problem, the mean squared error (MSE) between predicted and true values is usually used.

The training of a densely connected neural network is done with the back-propagation algorithm [23]. Given a batch (group) of training examples from a dataset, it requires computing output derivatives of all the layers, including the activation ones. The derivatives are then used to adjust the weights using various variants of the gradient descent algorithm.

The last piece required to train a neural network is the way to measure the quality of its results. Such a function is called a loss function. Again, it has to be a differentiable function, hence it is impossible to simply use the accuracy. In most regression cases, MSE can be used. It is expected that during the training the MSE between the predictions and values from the dataset (called 'reference', 'ground truth' or just 'true') will decrease. In the case of classification, the function used most frequently is cross-entropy, which, in simple words, decreases when the similarity between two probability distributions (predicted and reference) increases.

One of the most frequent problems when training a network is overfitting. When it happens, the network, instead of learning how to solve a problem, learns the expected outputs for given data but loses the generalisation ability, hence it performs poorly for new data. To detect overfitting, a dataset is divided into training and test sets. Only the examples from the training set are used for training, while the test set is used to validate the generalisation capabilities. Usually, when overfitting occurs, the loss function values for the training dataset still decrease while the loss function values for the test dataset start to drastically increase.

For the sake of avoiding bias towards the test dataset, the training dataset can be split again into the

final training set and a validation set. In such cases, the validation set is used to assess the quality of chosen models[2], and the test set is used only at the last stage to compute the final effectiveness of the solution.

## 2.4. Optimisers

Neural networks are trained using an algorithm called 'optimiser' which, in practice, is one of the gradient descent algorithm variants. Gradient descent is an iterative algorithm that uses the gradient of a loss function to adjust the weights of a neural network. An iteration of this algorithm is often called an epoch. In the base version, gradient descent can be expressed with the following formula:

$$\phi_{i+1} = \phi_i - \gamma \nabla f(\phi_i)$$

where $\phi$ is the set of parameters (in the case of neural networks – weights) either at iteration $i$, or at the next iteration $i + 1$, and $\nabla f(\phi_i)$ is the gradient of a loss function $f$. $\gamma$ is a learning rate – a hyperparameter[3] responsible for adjusting the training speed. A bigger learning rate might improve the training time, but, on the other hand, might also cause an inability to reach the minimum. The gradient is calculated with respect to the true values and the values predicted by the network. It is expected that after a given number of iterations, the loss function values will reach a minimum, giving the best quality of the network results.

The base version of the gradient descent algorithm has a lot of drawbacks. There are numerous variations fixing chosen drawbacks of the base algorithm. Nowadays, the one used most frequently is the Adam algorithm [24]. Its improvements include adjusting the learning rate dynamically or using a separate learning rate per parameter instead of the global one.

Usually, instead of processing the whole dataset at once, it is processed in batches (also called mini-batches) – groups of examples from the dataset. An optimiser following this approach is called 'stochastic with batches'. Most datasets are too huge to fit in the memory of a single GPU, so they have to be split into batches. Training the network using the stochastic version of gradient descent causes some difficulties since the gradient depends only on one batch at a time and might not lead straight to the loss function minimum. Ultimately, however, stochastic training is the only possible option considering the memory requirements. What is more, it is often suggested not to use too large batches, as the training might benefit from the smaller size [25, 26].

## 2.5. Convolutional layers

Dense networks are a reliable solution for the analysis of unordered input data. They process every pair of input features with the same importance. However, the order matters in some problems. For instance, in image processing pixels cannot be shuffled randomly. Usually, two neighbouring pixels have a closer relationship than two pixels on the opposite sides of an image. While a single neuron in a dense layer can learn to take only selected pixels into account, it is not efficient because

[2]Model refers to a specific neural network type (with a selected number of layers, neurons, etc.) or any other ML algorithm.

[3]There is a difference between hyperparameters and parameters. Hyperparameters (learning rate, number of layers and neurons, etc.) are set manually before training, whereas parameters (weights) are optimised during the training.

the neuron still has to store weights for the whole input. What is more, dense layers have a naive approach to dealing with translations of objects in an image. They can learn to recognise a flower in the middle of an image, but they will fail if the same flower is shifted to the corner. These drawbacks also apply to the one-dimensional counterpart of an image – time series.

The convolutional neural network (CNN) has been devised to overcome the above drawbacks. It is composed mainly of convolutional layers. A convolutional layer divides the input into small, usually overlapping fragments called receptive fields. Every neuron in a convolutional layer takes one receptive field as input and outputs a single value which can later be passed to an activation layer. Even though each receptive field is considered separate, all of them share a set of weights within a convolutional layer. In fact, a convolutional layer can have multiple weight sets called filters or kernels, but then each set is used for a separate output channel. A channel refers to a layer of an image or time series. For instance, in the 2D domain, the input often has three channels used to represent the RGB system. In the following layers, each channel stands for a different high-level feature, e.g. specific shape. The overall filter size is the size of a receptive field (width in 1D or width times height in 2D) times depth, i.e. a number of channels. Each filter is expected to learn some higher-level feature, no matter where it is located in an image. That is why usually the filter stride equals one, which means that the field is shifted by one sample. The concept of the receptive field and the stride is illustrated in Figure 2.4.

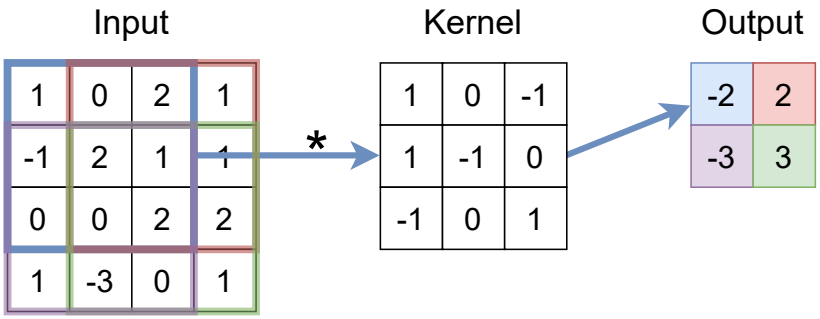


**Figure 2.4:** A convolutional 3×3 kernel (receptive field) applied to a 4×4 image (input) with a stride of one produces a 2×2 output.

The convolutional layer itself still does not fix the problem of translation causing a shift in the neuron activations. This problem is handled by pooling layers which collect the outputs of several convolutional neurons. This is usually done by pulling a maximum value from a receptive field with the size of 2 or 2×2 in the case of image processing (with different dimensions to the convolutional fields) with the stride equal to 2. The image dimensions are reduced twice after such an operation (see Figure 2.5).

Blocks containing convolutional and pooling layers can be stacked together so that the output of the first block is the input of the second one, etc. The blocks deeper in the network learn higher-level features than the first blocks and analyse bigger image chunks due to the max pooling. Sometimes, instead of using the max pooling in the last block, the GAP (Global Average Pooling) layer is used. It takes an average across every channel. This is particularly useful in the case of classification when the position of an object in an image is not relevant. Usually, a regular MLP is placed after a

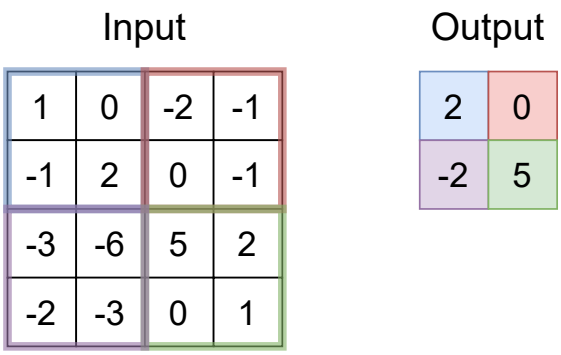


**Figure 2.5:** Example of max pooling with a 2×2 window and the stride of 2

bunch of convolutional blocks after the high-level features have been extracted. Thanks to that, the MLP can operate on unordered features extracted from the input.

## 2.6. Convolutional architectures

Nowadays, the most regular convolutional architectures are based on VGG [27]. Contrary to predecessors, it uses only small filter sizes: 3×3 or 5×5. Every convolutional block contains from two to four convolutional layers with the activation layers – most often ReLU – attached. Max pooling is applied after each block. The main assumption behind this architecture is that the convolutional network efficiency increases with its depth rather than the used filter size. The default task of a VGG network is classification. Therefore, an MLP is placed after the convolutional part in order to predict the probabilities.

Unfortunately, the depth of a VGG network cannot be increased forever. At a certain point, the performance increase is halted and the quality of results starts to decrease. One of the reasons is the problem of vanishing gradient. Its cause lies in the characteristics of the back-propagation algorithm and the fact that, for a deeper network, the gradient might become extremely small which drastically decreases the training perspectives.

The problem of vanishing gradient has been mitigated with the ResNet architecture [28]. It introduces residual blocks which employ so-called skip connections. Instead of a classical path of convolutions and activations, the outputs of the convolutional block are summed with unmodified input which ‘skips’ the convolutional layers. Among other pros, this improves the gradient flow and increases the reasonable network depth. Apart from that, convolutions with the kernel size of 1 are used at some points to decrease the layer depth, and, in turn, the number of parameters in the following convolutional layers.

While the network architectures described above are optimal in many classification or regression cases, a better architecture can be used for the annotation problem. Annotation is a regression problem of finding one or many points in the input image or time series. The problem discussed in this thesis is a perfect example – the goal is to find, in other words, annotate a peak in a time series. A 2D example could involve marking certain objects, e.g. animals, in an image. The primary example of an architecture devised with the above problem in mind is UNet [29].

The output of UNet is of the same shape as its input, hence it is an example of an autoencoder which encodes the input but preserves its shape. The input characteristics are the same as in the architectures mentioned above – it can be an image or a time series. The output, however, represents a heatmap where high values stand for the high probability of the annotated object being in a particular place. This requires preprocessing the dataset so that the ground truth values are expressed in the form of heatmaps, too. The heatmaps can be created in many ways. One of them involves applying a Gaussian on the annotation point [30]. In the case of predicting a particle timestamp, the ground truth vector contains the values of a Gaussian with the mean at the timestamp value and a chosen standard deviation.

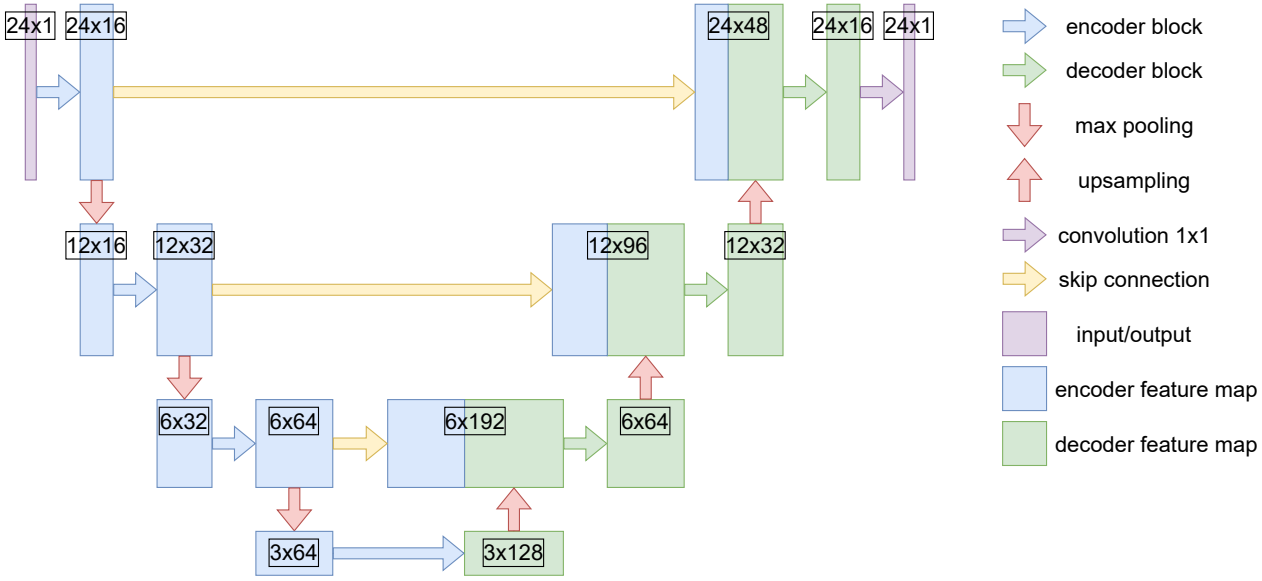


**Figure 2.6:** UNet architecture with example feature map dimensions for a time series with the length of 24 at input.

The network architecture is divided into an encoder and a decoder. The encoder's goal is to extract relevant features from the input, similar to a regular convolutional network. The decoder takes these features as input and predicts the output in the shape of the original input. Instead of regular convolutional blocks, it uses deconvolution blocks, which increase the time series length, so that it can be scaled back to the original size. Deconvolution can be accomplished by using upsampling which increases the size, and regular convolution, usually with the kernel with the size of 1 which reduces the number of channels. The UNet architecture for the time series processing is illustrated in Figure 2.6

## 2.7. Recurrent neural networks

Another network type commonly used for natural language and time series processing is the recurrent neural network (RNN). Instead of analysing a whole fixed-size input at once, it traverses the time series starting from the first sample and memorises relevant features from the samples processed so far. Therefore, it is capable of having a time series of any length at input. An RNN, apart from computing the prediction for each input sample, computes the hidden state which is not outputted but given as input for the next sample.

In its base form, an RNN considers only the most recent hidden state. It might not suffice, as sometimes, especially in the tasks of natural language processing, it is important to remember the state from many words ago. The LSTM (long short-term memory) [31] unit has been invented to enable keeping all the relevant information from the past. A single LSTM unit contains a bunch of gates that steer which parts of the hidden state to keep and which to overwrite or modify for the next time step. The gates mix the hidden state and current input in order to provide a prediction which is based both on the current sample and on the past. Another, newer, but less complex solution is the GRU gate [32].

RNNs can be used to predict a single number. In this case, the prediction can be overwritten at each time step. Another possibility is to predict a heatmap as UNet does. Each heatmap sample is then predicted when the network processes the respective input sample. To increase the efficiency, an RNN unit can predict many channels. In such a case, each channel is predicted with a separate set of parameters. An MLP can be used afterwards to use the features predicted by the RNN and transform them into the output of expected shape. Finally, RNN units can be stacked together, so that the output from one is the input for another. In that case, the RNN layer works across many channels – each output channel is based on the respective sample from every input channel.

## 2.8. Other layer types

Among others, there are three interesting layer types that can be used in certain cases to improve performance.

The first of them is the dropout layer [33]. It randomly zeroes a selected number of neurons in each layer to mimic ensemble learning. In ensemble learning, many models are trained together to increase the overall efficiency. Dropout assumes, that in each training iteration, a different network is trained since different neurons are deactivated. Dropout enables using much bigger architectures without overfitting. However, it has to be used wisely not to lose too much information.

There is a similar layer to dropout better suited for convolutional layers – spatial dropout [34]. While the regular dropout chooses the neurons to drop randomly for each convolutional channel, the spatial dropout drops the same neurons in all of the channels. This is supposed to promote independence between neurons.

The other layer is batch normalisation [35]. It normalises its inputs across every batch with the goal of making the training more stable by eliminating huge values inside the network. What is interesting, the reasons behind the success of this layer are not known precisely and are still under discussion [36].

# 3. Classical and machine learning approaches for the time of arrival computation

This chapter starts with a comparison of the classical offline[4] methods used in the prediction of the time of arrival timestamp based on a sampled time series. Afterwards, it describes the state-of-the-art deep learning approaches. Apart from the use cases in high energy physics (HEP), similar problems from other fields are analysed, too.

## 3.1. Classical approaches

Several classical approaches were compared in Chapter 4.3.2 of [37]. The methods were tested in a similar setup to the one discussed in this thesis, but the waveforms contained much more samples and were of higher quality. What is important, the waveforms included both rising and trailing edges of the signal, as opposed to the waveforms from the dataset used in this thesis which most often contain the rising edge and only a part of the trailing edge.

It was possible to find the resolution of a method because each event had two waveforms from different sensors of the same architecture. This allowed the computation of a timestamp difference for each event. The resolution was determined by computing the standard deviation of the distribution of the timestamp differences, divided by the square root of 2 ($\sigma_{\mathrm{T}}$). A detailed description of this method, including its improvement involving a Gaussian fit, can be found in Chapter 5.7. Selected results are presented in Table 3.1.

| **Algorithm** | $\sigma_{\mathrm{T}}[\mathrm{ps}]$ |
|---|---|
| Fitted normalised threshold | 221 |
| Offline CFD | 210 |
| Extrapolation | 196 |

**Table 3.1:** Comparison of selected classical methods. CFD refers to the offline implementation of the analogue algorithm. Source: [37]

The best resolution was achieved by the extrapolation method, but it included fitting a 6th-degree polynomial to the rising edge of the signal, and it was often challenging to reach the convergence. This task would be more difficult or even impossible in the case of this project because the waveforms are very diverse. The second best method was the offline implementation of the analogue constant fraction discriminator algorithm. Nonetheless, in the end, the author decided to use the fitted normalised threshold (FNT) algorithm. This method was described as the most reliable and capable of the same results as CFD if properly parameterised. The same relation was also mentioned in [16].

The FNT method included applying a fit with a second-order polynomial on the maximum of the signal to acquire the amplitude with higher precision. Then, it identified the rising edge as the signal

[4]The word ‘offline’ in this thesis is used to describe an algorithm working on an already prepared dataset, i.e. when the data have been acquired and, if needed, properly calibrated. Online algorithms, on the other hand, work in the software or analogue pipeline straight after the data acquisition.

between crossing 20% and 80% of the maximum amplitude and fitted a line to these two points. Finally, the standard normalised threshold algorithm was used, but the timestamp was computed on the fitted function. This approach benefited from the high quality of analysed waveforms. As described above, the PPS waveforms acquired in the LHC are much more difficult to analyse which decreases the potential gain from using fits. Also, in a sizeable fraction of the waveforms, the rising edge does not fit in the window leaving the maximum outside. This is another reason why fitting a polynomial around the maximum is not viable. Thus, it is the normalised threshold algorithm that is used for the official event reconstruction at the LHC, and it is also the reference method used in this thesis. The small adjustments are described in Chapter 4.

## 3.2. Deep learning for time reconstruction in the MRPC detector

Multi-gap resistive plate chamber (MRPC) is a gaseous detector used in time-of-flight systems. Such systems measure the same particle in two or more places and compute the time needed for the particle to travel between different sensors. Exactly like in the case of diamond detectors mounted in PPS, if connected to fast readout systems, MRPC can produce sampled waveforms of voltage. A peak in such waveforms means that a particle has been detected.

To start with, it was proven in [38] that a neural network can beat the time-over-threshold (TOT) algorithm. For this purpose, the readout electronics were changed to output a full, sampled waveform instead of just two samples needed by TOT. The neural network was an MLP with circa 5 layers, each containing around 15 neurons. It outputted the length of the signal's leading edge. The particle arrival time could be calculated as the difference between the peak time and the leading edge length. This is a non-obvious approach since the network could also simply return the time of arrival timestamp. In the end, the deep learning approach managed to achieve much better results than TOT (by around 40%). This article shows, that a neural network can be effectively used in the task defined in this thesis, however, its performance was not assessed with respect to CFD which would likely outperform TOT as well.

A similar MLP network was compared with a recurrent, LSTM network in [39]. The MLP was composed of 6 layers and each layer – of around 16 neurons. In the case of the LSTM network, its output was fed into a 2-layer MLP. As previously, the network was trained to output the length of the leading edge, given a waveform of 10 samples. The LSTM network turned out to perform better than MLP. Such a relation is expected for the PPS data, too. Again, the LSTM performance was compared with TOT. In this case, the gain was much smaller than previously, but it was still noticeable.

The work from two previous articles was extended in [40], in which the ComLSTM (Combined LSTM) network was proposed. The novel architecture combined LSTM layers with fully-connected ones. In such a network, the input is transmitted in two separate paths. Path A starts with an LSTM layer (420 units) and is followed by a 3-layer MLP (128/32/1 neurons). Path B starts with a fully-connected layer (100 neurons) which is followed by an LSTM unit (400 units). The path is completed with a fully-connected layer with a single output. The final output is the weighted sum of the outputs of the two paths. By using the ComLSTM network, the authors were able to satisfy imposed timing requirements. Unfortunately, the comparison between ComLSTM and other

network types was not presented in the article. Nonetheless, the article shows that mixing different network types can yield good results.

A two-path approach was also presented in [41]. In this case, however, waveforms from 8 readout channels were analysed at once. Moreover, the input to the first path was a whole 80-ns-long waveform, while only a 4-ns-long waveform containing the peak was given to the second path. This approach is not viable in this project, since the waveforms do not contain the whole peaks, and, most often, the baseline is present on just a few samples. However, this article compared the performance of the neural network with CFD, and the improvement was noticeable (around 20%).

## 3.3. Convolutional neural networks for time-of-flight estimation

The problem of predicting the signal timestamp is also present in medical appliances, such as Positron Emission Tomography (PET). The goal is, as previously, estimating the time-of-flight of a proton by analysing time series of voltage produced by the detector. In this case, however, two waveforms are given on input, and the goal is to estimate the time difference between them, which is a similar problem to just predicting the timestamp. Thus, if one solution is good at time-of-arrival prediction, it is expected to have a good performance for predicting only the time of arrival.

As an example, the authors of [42] proposed a convolutional neural network which was able to achieve better results than the leading edge discriminator (LED) and CFD. In this case, LED refers to the fixed threshold algorithm which fixes the time walk effect by utilising an analytical correlation. CFD means the offline implementation of the analogue algorithm. Two network architectures were evaluated. The first one was equipped with smaller convolutional filters of 5 samples, while the filter sizes in the other one were even up to 11 samples. These numbers are big considering the current state-of-the-art standard filter sizes which most often do not exceed the width of 3 samples. Both architectures were tested in configurations consisting of various numbers of layers. Eventually, their performance was comparable. This led the authors to an interesting discovery – the impact of the number of layers was much bigger than the impact of filter size or the number of feature maps. Hence, only small filters are considered in this thesis.

The other important conclusion with respect to this thesis is the application of a convolutional network. CNNs have been primarily devised for image processing, but time series can be considered one-dimensional images. This article proves this point of view, which makes CNN an obligatory network type for testing in this research.

A very similar problem is discussed in [43]. This one is even more relevant since it addresses the time-of-flight estimation in high energy physics using popular timing detectors called MCP-PMT (MicroChannel Plate PhotoMultiplier Tube). The authors used a simple convolutional neural network to improve the CFD time precision from 57.4 to 28.5 ps.

Finally, both articles use the same method of computing the precision as the one used in this thesis (see Figure 3.1). It involves plotting the differences between two values, applying a Gaussian fit and retrieving the standard deviation of the Gaussian (as described in Chapter 5.7), which is the measure of the precision.

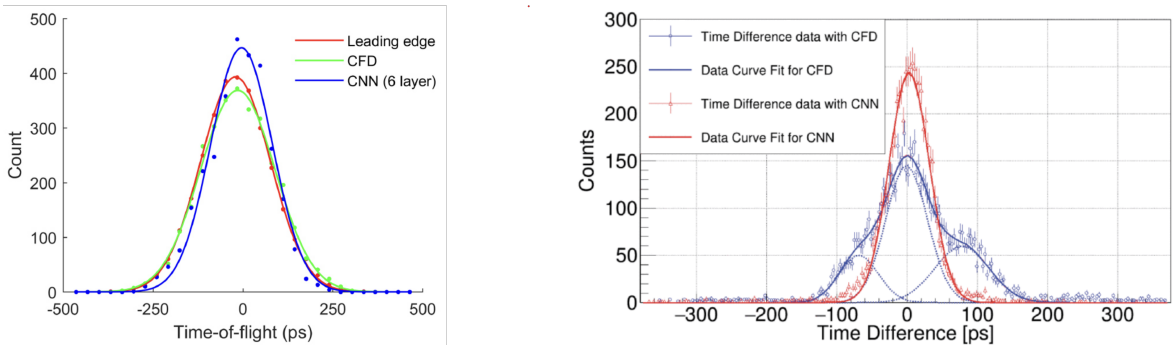


**Figure 3.1:** Time difference histograms proving the better time precision (narrower fitted Gaussian) of a neural network with respect to other methods. Source: [42] (left), [43] (right)

## 3.4. Neural networks for ECG

Electrocardiography is another domain of medicine which strongly relies on waveform analysis. Various disorders can be diagnosed by analysing electrocardiograms (ECG). This task can take even up to several days for a cardiologist, thus many machine learning approaches have been proposed. This problem seems to be much more complex than timing measurements in HEP. Nevertheless, the core of the problem is the same – to analyse the waveform and predict a number or a set of numbers.

One of the important tasks is the annotation of specific fragments of ECG, namely R peaks and P waves. A machine learning approach was considered in [44]. A CNN was used for this purpose. Its goal was not to predict the places in which the peaks or waves occurred, but to annotate a waveform, or in other words, classify it. A method to predict the timestamps was devised anyway, but it required other algorithms for finding the extremums. The network architecture was typical – there were three convolutional blocks, each containing a convolutional layer with a kernel size of 3 and an increasing number of filters. The block employed batch normalisation layers, too. The activation function was PReLU [45] instead of the most commonly used ReLU. Also, dropout was used not only in the MLP at the end of the network but also after the convolutional and pooling layers, which is not typical.

Even though regular CNNs achieve good results in prediction, they have primarily been devised for classification. Convolutional layers can be used in more sophisticated architectures, such as UNet which seems to be one of the best-suited architectures for annotating certain places in a waveform. Such an approach was employed in [46]. This article focuses on the analysis of waveforms acquired from wearable devices. Such waveforms are of poor quality and contain a lot of noise, which is especially intriguing taking into account the noisy nature of the waveforms in this thesis. The authors devised U-shaped convolutional networks with skip connections and 6 layers both in the encoder and the decoder part. Again, batch normalisation layers were placed after the convolution ones. The network was able to outperform all state-of-the-art methods, including the LSTM-based models, which are also tested in this thesis. Other, non-ML approaches were also considered, but the methods used in HEP timing measurements, such as CFD, were not included.

A similar network architecture was presented in [47] and named RPAA (R Peak Annotation Algorithm). Even though the U-shaped architecture was kept, each convolutional block was much

more complex and included splitting the input for a few separate paths. Due to the smaller problem difficulty and already testing numerous neural network architectures, such a strategy is not used in this thesis. However, the thesis might benefit from the data representation method. The network from the previous article outputted a 1-D segmentation map – for each sample in the waveform it was either 0 (no peak) or 1 (peak detected) which only enabled predicting a timestamp on a particular sample. In the case of this article, the network was trained to predict the distance to the closest peak which allowed marking a peak between two neighbouring waveform samples. Since the waveforms in this project have only 24 samples, the prediction cannot be limited by the sampling step.

Using the distance-based data representation is not common in the field of annotating an image or a time series. Applying a Gaussian with a small standard deviation to the timestamp (1D) or keypoint (2D), as in [48], is done much more often. Nonetheless, both approaches are tested in this thesis

# 4. PPS dataset preparation and characteristics

This chapter characterises the source, preprocessing and final form of the main dataset used in this thesis. Other, similar datasets are also employed in further chapters, but they are produced in the same way. The only difference is the input LHC data used. As CFD is vital during the data preparation phase, its implementation is also described in detail.

## 4.1. Data source

The LHC is not operating nonstop. It can either host and accelerate particle bunches or be empty. A period between the injection of particle bunches and dumping them is called a fill and usually lasts between a few hours and one day. Various LHC conditions can be modified between fills. Small changes often occur even during a fill. To distinguish the periods with the same conditions, a fill consists of one or several runs[5]. The raw data used to build the dataset used in the main numerical experiment of this thesis comes from the LHC run 355207 which took place in 2022. This run was selected arbitrarily among other runs that collected SAMPIC data in 2022. A 2023 run with SAMPIC data was also available, but it was not used in the main experiment as waveforms collected in 2022 are much clearer and better for analysis compared to 2023. Nonetheless, a 2023 run and other runs from 2022 were also used for secondary experiments to verify the generalisation of the networks.

The entire LHC ring is divided into several sectors. The sectors adjacent to the CMS IP are sector 45 (anticlockwise of CMS when looking down at the LHC ring) and sector 56 (clockwise of CMS). In 2022, PPS employed one timing RP per sector. However, in 2023, the number of RPs in each sector was upgraded to two [49]. Nevertheless, the main dataset used in this thesis was built from data collected in a 2022 LHC run, which used only the RP in sector 56. The waveforms collected in 2022 using the RP in sector 56 are of better quality and thus more suitable for analysis than data from sector 45. Signals from sector 45 were also used in secondary experiments to measure the generalisation capabilities, similar to using other runs.

A PPS timing RP hosts four planes, as shown in Figure 1.2. Every plane contains 12 double-diamond sensors. Each sensor is connected to a separate readout channel. The channels are numbered from 1 to 12, starting with the ones closest to the LHC beam. Since every plane has the same architecture and the sensors are located in the same places, in perfect conditions a particle flying through an RP should be detected four times. Sometimes, however, a particle can be measured fewer or even more times. This can happen when it passes directly between two channels. If the signal is strong enough, the particle is detected in both channels. Otherwise, it is detected either once, or not at all. As described in the first chapter of the thesis, only channels 2 and 11 of each plane are connected to the SAMPIC readout device. Hence, only data from these two channels were used in this research. The channel layout and the most common particle traces registered by the detector are depicted in Figure 4.1. It is worth noting that RP channels are often indicated with two coordinates. Channel (X, Y) stands for channel Y of plane X.

[5]This thesis follows the popular convention of using the noun ‘run’. The capitalised Run stands for a running period of the LHC (for example, Run 2 lasted from 2015 to 2019 and Run 2 started in 2022). The uncapitalised run stands for a part of a fill.

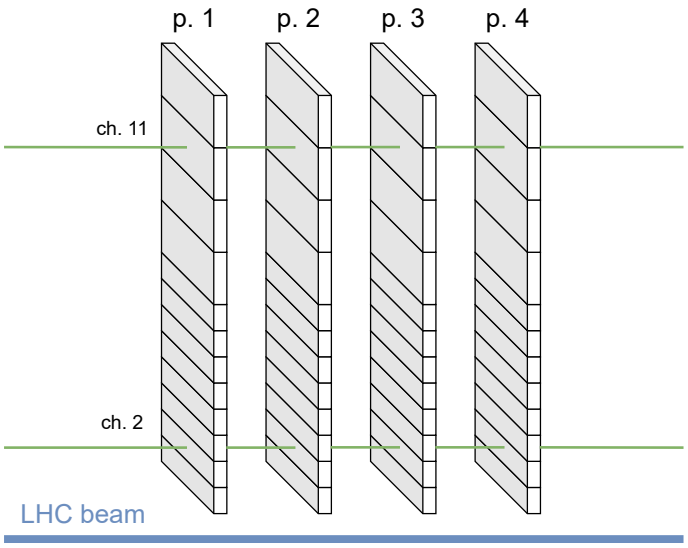


**Figure 4.1:** Layout of planes (p.) and channels (ch.) in a single detector package (RP). Most common particle traces are marked in green. Each grey cuboid represents a double-diamond sensor. Channels are numbered from 1 to 12 starting with the one closest to the LHC beam. Only channels 2 and 11 are connected to SAMPIC, so only data from these channels are analysed in this thesis.

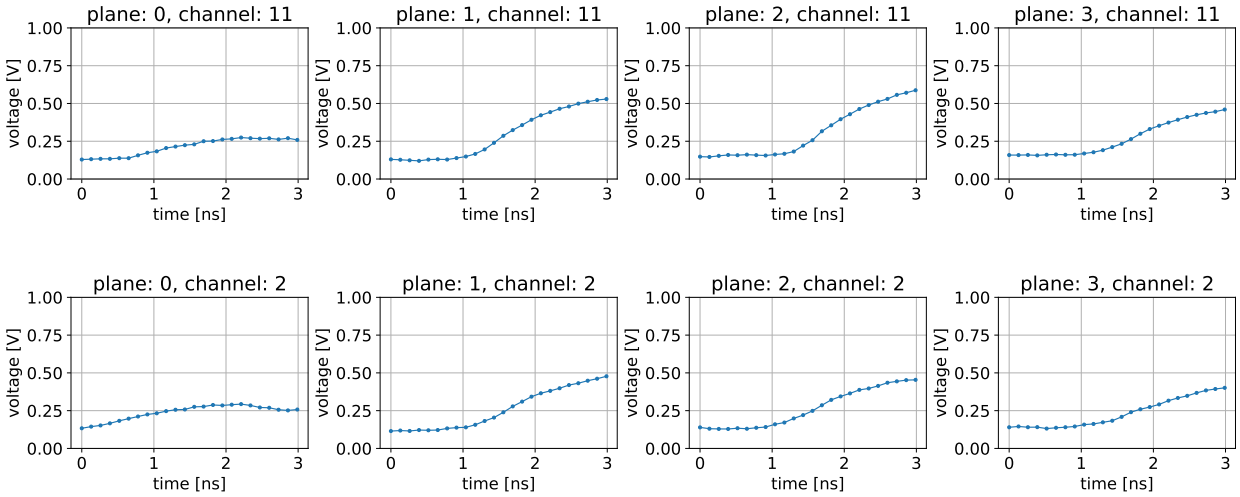


**Figure 4.2:** Waveforms from two representative events measured with all four planes

The raw dataset was built on the data already preprocessed in the CMS software framework (CMSSW). It contains a list of events. An event corresponds to measurements of a single particle in various readout channels. The data are already grouped by the detector plane and channel. A single entry contains a waveform and $t_0$. The waveform is a time series of 24 voltage samples, so it contains two vectors – one with 24 voltage values, and the other with 24 time values in nanoseconds starting at 0 ns. The time vector is redundant since the time step between two samples is the same for all the data and equals circa 130 ps. $t_0$ stands for the time of the first measurement, i.e. sample, in the waveform. $t_0$ is needed during the preprocessing stage to find a relative time shift between the

particle measurements from different channels. Example waveforms are shown in Figure 4.2.

Before beginning the preprocessing, it is helpful to count the number of channels which detected the particle for each event and plot them in a histogram called the hit count histogram. It is shown in Figure 4.3. The histogram suggests that one of the planes might have been not fully operational. Otherwise, four hits would be the most common histogram entry. Note that signals from plane 0 in Figure 4.2 are weaker than others, which implies that this might be the malfunctioning plane. At this point, this is just guessing, but this issue is deeply analysed in the following parts of this chapter.

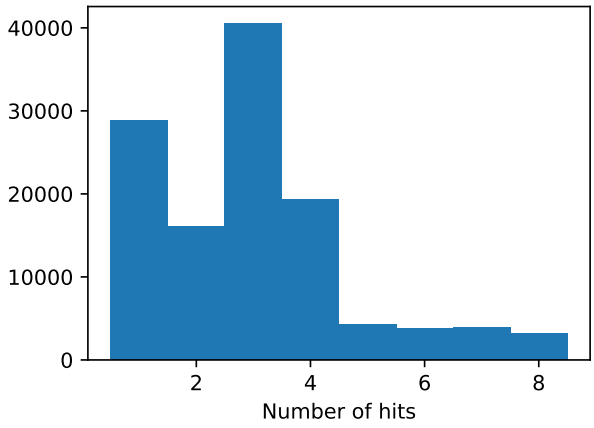


**Figure 4.3:** Hit count histogram right after loading the input dataset (before preprocessing)

## 4.2. Constant fraction discriminator

The constant fraction discriminator (CFD) has its roots in an analogue device used to minimise the impact of the time walk effect. The CFD device copies the signal and then inverts, attenuates and delays it. Afterwards, the new signal is summed with the original one. The zero-crossing timestamp of the resulting signal does not depend on the signal amplitude anymore.

There is no need to employ such a device in this work. The signal is sampled using SAMPIC, so the whole analysis can be done offline using more accurate and flexible methods taking a digitised waveform as their input. Therefore, in this thesis, CFD is implemented in an offline fashion, using the normalised threshold algorithm with several tweaks.

The algorithm itself is straightforward (see Figure 4.4). At first, a baseline is subtracted so that each signal starts at around 0 V. The baseline is calculated as the mean of the first six samples. The number of the first samples was chosen experimentally. The goal was to take as many samples as possible to increase the baseline estimation accuracy. On the other hand, the number had to be small enough not to include the start of a voltage peak caused by a particle. This can be verified by plotting a histogram of baseline samples (the first six samples from each waveform). The baseline samples are expected to contain only random noise, thus the histogram should resemble a Gaussian. Figure 4.5, left, shows that the histogram created with six samples from every waveform is symmetrical. On the other hand, if more samples are included (Figure 4.5, middle and right), the histogram becomes unsymmetrical with the right tail representing the beginning of a signal.

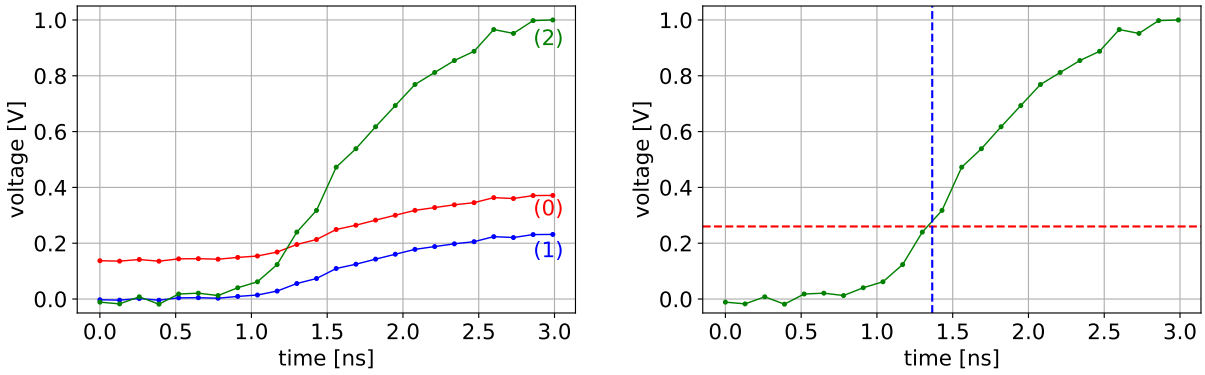


**Figure 4.4:** Depiction of the implementation of the CFD algorithm. Left: data processing steps: (0) raw waveform, (1) baseline subtraction, (2) normalisation – final waveform; right: applying a selected threshold (red dashed line fixed on 0.26 V) and retrieving the timestamp value as the moment of crossing the threshold (blue dashed line).

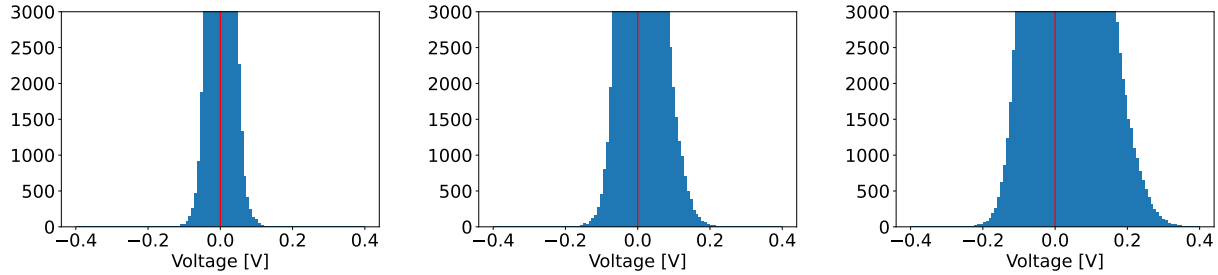


**Figure 4.5:** Histograms of baseline samples for baselines composed of first 6 (left), 8 (middle) and 10 (right) samples. To compensate for different baselines and amplitudes, the baselines were subtracted and the signals were divided by their amplitudes before plotting. While the first histogram (6 samples) is symmetrical (with the mean around 0 V represented by the red line), the middle and right ones have a right tail indicating including too many samples in the baseline calculations. The histograms were cut at the maximum value of 3000 to increase the visibility of the right tails.

The next step is normalisation – each sample is divided by the maximum value of the waveform which brings its amplitude to 1 V[6].

After these two steps, every waveform starts at around 0 V and has an amplitude of exactly 1 V. Because the amplitude is constant, the fixed threshold algorithm can be employed. The final timestamp value can be retrieved as the moment of crossing a selected threshold. To find the exact value, the linear fit is applied between two samples closest to the crossing point – the first one is the last sample with a voltage smaller than the selected threshold. The other sample is the next one – the first sample with a voltage value bigger than the threshold. After that, the crossing timestamp can be easily found using the linear equation.

[6]This thesis uses the convention in which even after the normalisation the time series values are given in volts (V) to increase the readability.

This method is not perfect. Due to the noise in the data, the normalisation is not flawless, and the small impact of the time walk still persists. Nonetheless, it is simple and does not require a lot of expert knowledge and a meticulous parameter optimisation process. Its only parameter is the threshold. It cannot be chosen randomly, however. It should be as small as possible since the time difference caused by the time walk effect is smaller when a small threshold is used (see Figure 1.4). On the other hand, the threshold cannot be too small to avoid reaching the threshold by fluctuations caused by the noise. Choosing the optimal threshold is one of the preprocessing steps described in the next part of this chapter.

## 4.3. Preprocessing

The preprocessing stage can be divided into a few steps described below.

### 4.3.1. Removing saturated and noisy events

The goal of the first step is to filter out unwanted data. Noisy events are the waveforms in which a particle signal cannot be distinguished from noise. In saturated events, the voltage exceeds the readout capabilities of the detector which leads to capping the voltage at a certain level. As a consequence, a true amplitude cannot be retrieved from the waveform, and the time walk effect cannot be properly handled by CFD. The examples are shown in Figure 4.6.

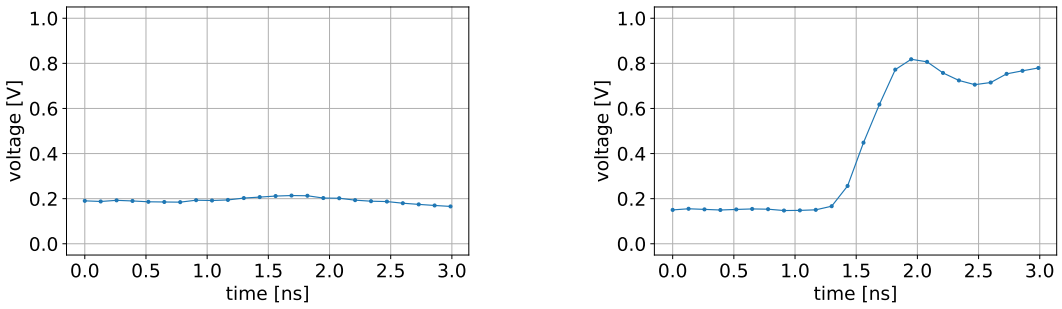


**Figure 4.6:** Representative examples of noisy (left) and saturated (right) events. The bump in the right signal is common among saturated data – usually, the signal reaches the maximum, bumps down for a moment, and then comes back the high voltage again.

Filtering out these events is done by plotting the amplitude histogram separately for each channel. Noisy signals are characterised by a small amplitude so they are located on the left side of the histogram. The saturated events reach the readout maximum so they are on the right side of the histogram. Usually, those two groups form two peaks so they can be easily separated from the healthy events. The filtering is presented in Figure 4.7.

The plane 0 histograms indicate the reason for anomalies related to the hit count histogram described earlier in this chapter. The plane or its readout was not operating properly during the chosen run, and most often only noise was measured. Therefore, it was decided not to include plane 0 in further processing. Consequently, from now on, only the data from planes 1, 2 and 3 is analysed.

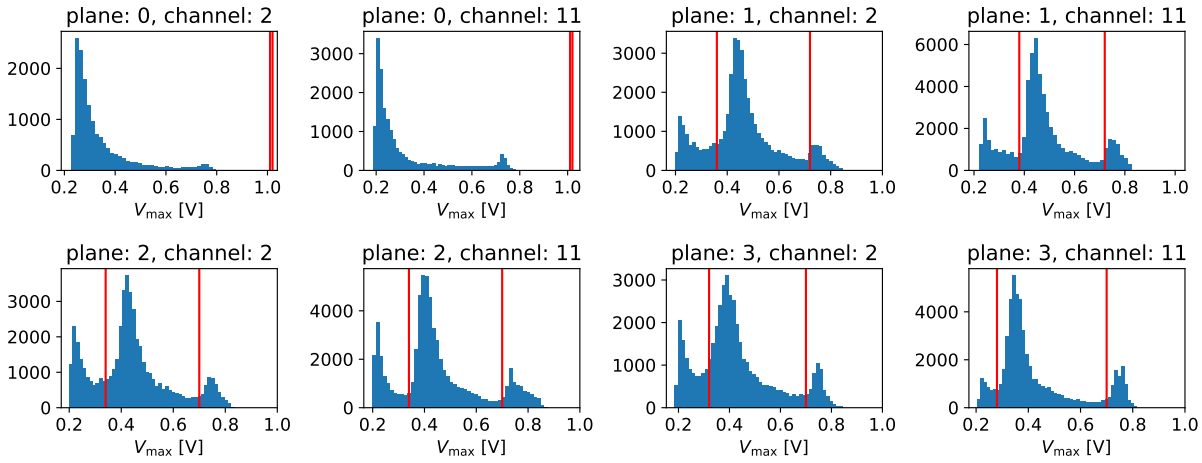


**Figure 4.7:** Maximum voltage (amplitude) histograms for each readout channel. Only events between the red lines are kept for further processing. The filter values were chosen experimentally for each channel to limit the probability of including unwanted events and not filter out too many good events. Due to too much noise present in the data from plane 0, filters are placed on values bigger than 1 to filter out all the events.

While saturated events were filtered out from the main dataset, they are analysed in this thesis in one of the secondary experiments. CFD does not perform well on these events, but ML algorithms might be able to retrieve the amplitude even from a capped signal.

### 4.3.2. Removing wrong first sample times

As mentioned earlier in the chapter, the first sample times ($t_0$) are provided together with the waveforms. Plotting a histogram (shown in Figure 4.8) revealed that some of them are outliers, i.e. lie far from the rest of the timestamps. They were filtered out to avoid a decrease in the quality of the dataset.

### 4.3.3. Finding the optimal CFD threshold

To build the dataset, it is required to compute the CFD timestamp for each waveform (explained in Chapter 4.4). Choosing the optimal CFD threshold is an essential step in increasing the dataset quality. To simplify the process, it was decided to use the same threshold for the whole dataset. However, computing the optimal threshold for each channel could bring a slight improvement.

The procedure is done by finding the minimums of two metrics (described below) for various thresholds. The metrics rely on finding the standard deviations of the differences between the CFD timestamps of two waveforms from the respective channels from a pair of planes (for instance, channel (1, 11) and (2, 11)). Contrary to the standard deviation, the mean of the differences cannot be used here, as it depends on the readout shift between channels. Such a constant shift between two channels is inevitable and stems, for instance, from the differences in the cable length. In a perfect scenario, the difference between two channels would be constant and equal to the channel shift. However, due to the noise, hardware differences, and flaws of CFD, the difference distribution is smeared and resembles a Gaussian (similarly to the histograms in Figure 4.13). The goal is to find

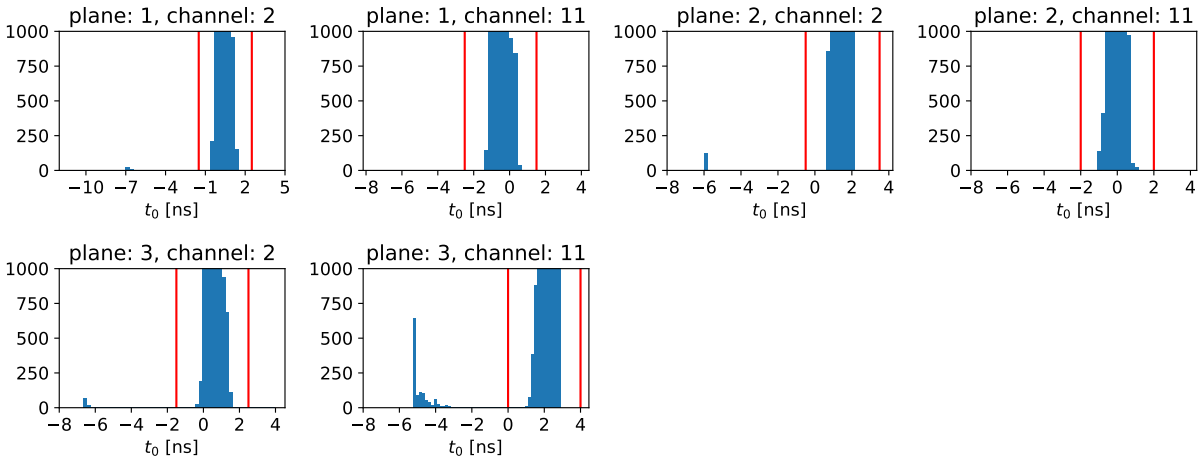


**Figure 4.8:** $t_0$ histograms for each channel. Only events between the red lines are kept for further processing. The filter lines were chosen so as to keep only the main peak on the histograms. The histograms are cut at the maximum value of 1000 to increase the visibility of outliers.

such a CFD threshold, that the difference distributions are as narrow as possible, i.e. the standard deviation is the smallest. The final metrics take into account the average and maximum of the channel pairwise difference standard deviations (i.e. the difference standard deviations for each pair of respective channels).

The metric computation can be summarised in a few, looped steps below.

1. Choose a new threshold value.
2. Compute the CFD timestamps.
3. For each pair of channels, compute the standard deviation of timestamp differences.
4. Compute two metrics – average and maximum standard deviation (from the values computed for each pair of respective channels).

The metric values for different thresholds are shown in Figure 4.9. The paramount goal is to avoid computing the CFD timestamps based on noise. Therefore, the maximum of difference standard deviations is considered more important. The minimum of this graph is for the threshold of 0.2, hence this was the threshold used in the experiments and analysis.

It can be also verified if the chosen threshold is not too small, i.e. is not taking noise into account. The baseline histogram (Figure 4.5, left) can be reused for this purpose. A safe minimum for the CFD threshold is the standard deviation of the baseline points multiplied by 5 which equals $0.016 \cdot 5 = 0.08$ in this case. Thus, the CFD threshold of 0.2 is perfectly safe.

### 4.3.4. Computing CFD timestamps and adjusting first sample times

Now that the optimal CFD threshold has been chosen, the CFD timestamps can be computed for every waveform using the optimal threshold. The next step is to adjust the first sample times ($t_0$).

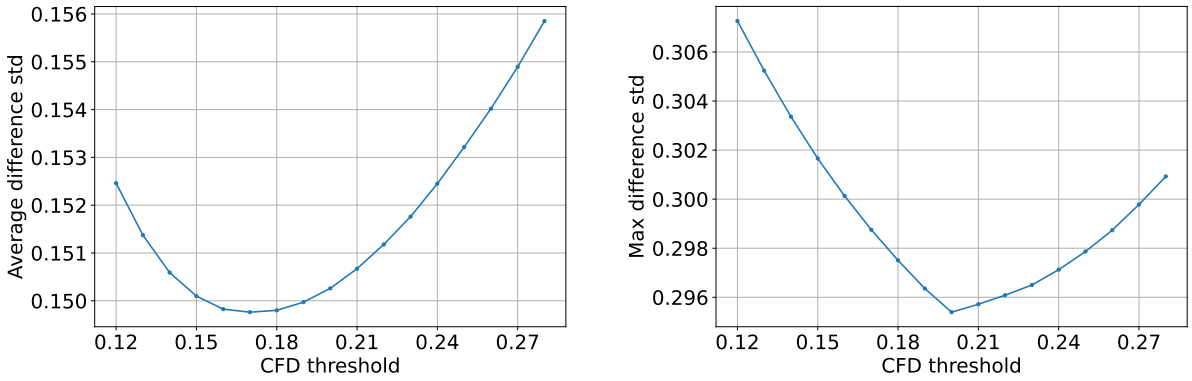


**Figure 4.9:** The CFD threshold metrics computed for different CFD thresholds

Base $t_0$ is, in fact, the time of trigger, which stands for the time when an event was detected and saving the voltage samples was triggered. Unfortunately, it does not take into account the readout characteristics and cable length. Finding the estimation of the exact first sample time is relatively straightforward. The goal is to find the timewise relation between the measurements of particles in different channels. Thus, the global, nonrelative timestamp is not required. Only a timestamp relative to other channels is needed.

This can be achieved by calculating the sums of the CFD timestamp and $t_0$ for each event in a particular channel and plotting these sums in a histogram. The average sum of the CFD timestamp and $t_0$ should be the same for each channel. The most straightforward choice is to bring the averages to zero. It can be done simply by subtracting the average from the original $t_0$ for each event[7]. The histograms before the subtraction are shown in Figure 4.10.

### 4.3.5. Taking only the events with three plane hits

Since it was decided to process only the events with exactly three plane hits, the dataset used in further steps contains only the events in which a particle was measured by planes 1, 2 and 3 (exactly once by each plane). The filtering can be depicted by plotting the hit count histograms before and after preprocessing. They are displayed in Figure 4.11.

## 4.4. Final dataset

The last piece needed to construct the final dataset is producing ground truth timestamps, i.e. reference timestamps a neural network is expected to learn while taking a waveform as input. This is why from the neural network perspective, they are 'true', and they are often called 'ground truth'. The reference timestamp cannot be just the CFD timestamp. The network performance is limited by the training dataset, so the model would be only capable of reaching the same performance as CFD. Therefore, the true timestamp is calculated as an average of the CFD timestamps of the same particle from all the other channels. For instance, the ground truth timestamp for a waveform measured with channel (1, 2) is the average of the timestamps from channels (2, 11) and (3, 11). The

[7]From now on, the first sample time ($t_0$) refers to the adjusted $t_0$ (original $t_0$ including the subtraction of the histogram average).

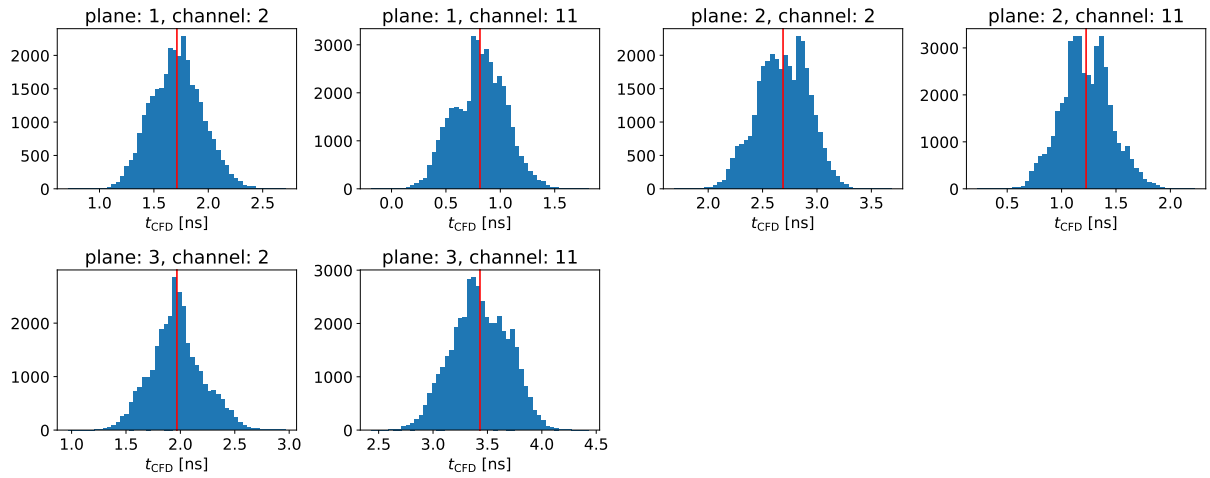


**Figure 4.10:** The histograms of the CFD timestamp and the respective $t_0$ sums. Averages are marked with red lines. The only change caused by the preprocessing step described in Chapter 4.10 is shifting these histograms in such a way that their averages (red lines) are equal to zero.

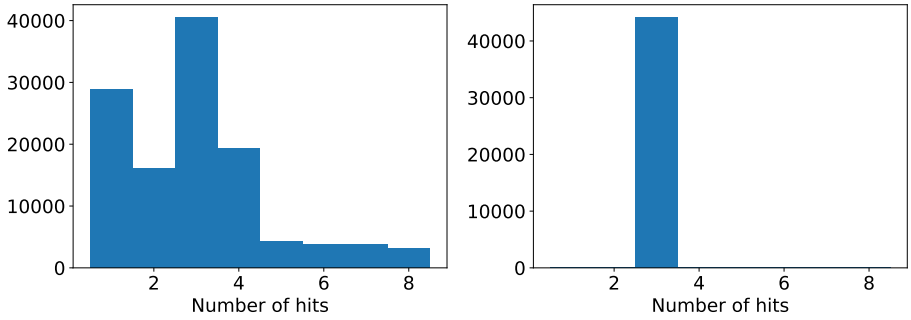


**Figure 4.11:** Hit count histograms before (left) and after (right) filtering only the events with three plane hits.

timestamp from plane 1 is not included in the calculation, since it would cause a bias in favour of CFD. If it was included in the ground truth calculation, the CFD timestamp would be closer to the expected result causing an unjustified increase performance boost of CFD.

Having computed the true timestamps, it is useful to plot them on a histogram for each channel. As shown in Figure 4.12, most timestamps lie between 0.5 ns and 2.0 ns. However, 98 events lie outside this range. They are filtered out as outliers. The ground truth timestamps can be also validated by plotting the histograms of differences between the CFD timestamps and the ground truth timestamps. They are shown in Figure 4.13. The histograms resemble Gaussians centred in 0 which was expected as the error is random.

It is worth mentioning, that $t_0$ is included in the calculation of timestamp averages. However, the final timestamp stored as the ground truth value is brought back to the domain of a specific channel, i.e. it is relative to $t_0$ which, in turn, means that it specifies the time from the time series beginning. Thanks to that, there is no need to put $t_0$ into the final dataset. Examples of waveforms and their

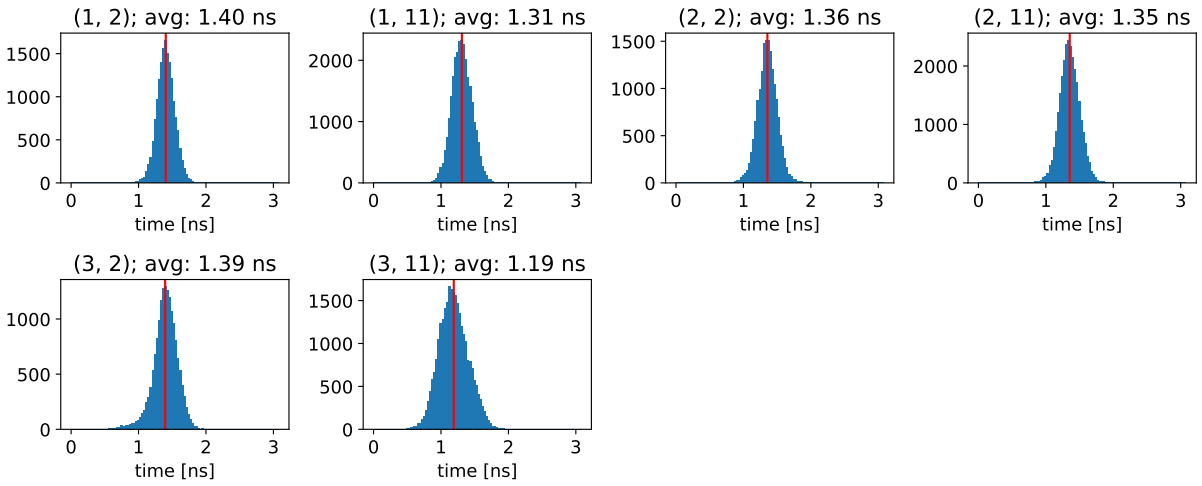


**Figure 4.12:** True CFD timestamp (ground truth) distributions for each channel with averages marked in red

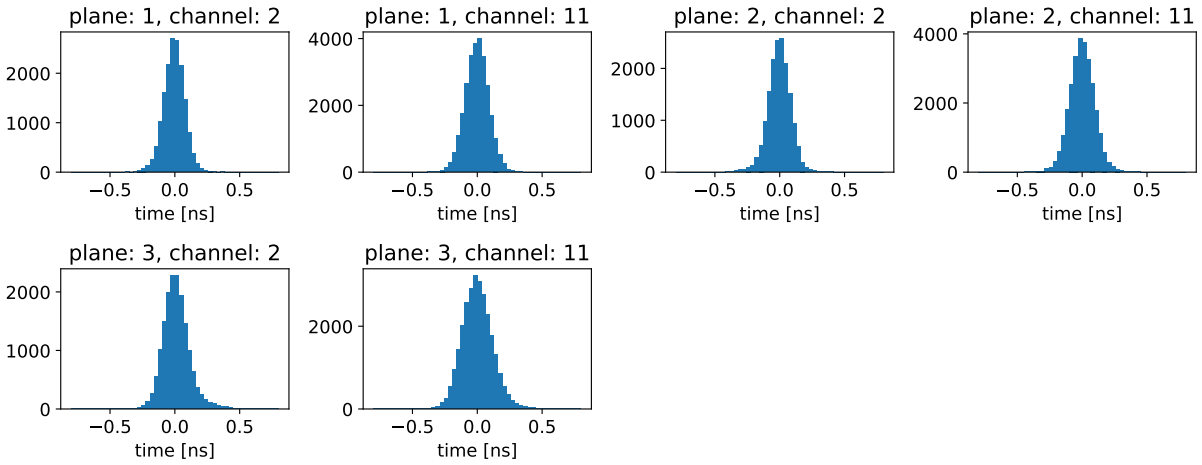


**Figure 4.13:** Histograms of differences between CFD and ground truth timestamps

respective ground truth timestamp values are shown in Figure 4.14.

The final dataset contains:

- 16395 signals produced by plane 1, channel 2,
- 27786 signals produced by plane 1, channel 11,
- 16513 signals produced by plane 2, channel 2,
- 27668 signals produced by plane 2, channel 11,
- 16399 signals produced by plane 3, channel 2,
- 27782 signals produced by plane 3, channel 11.

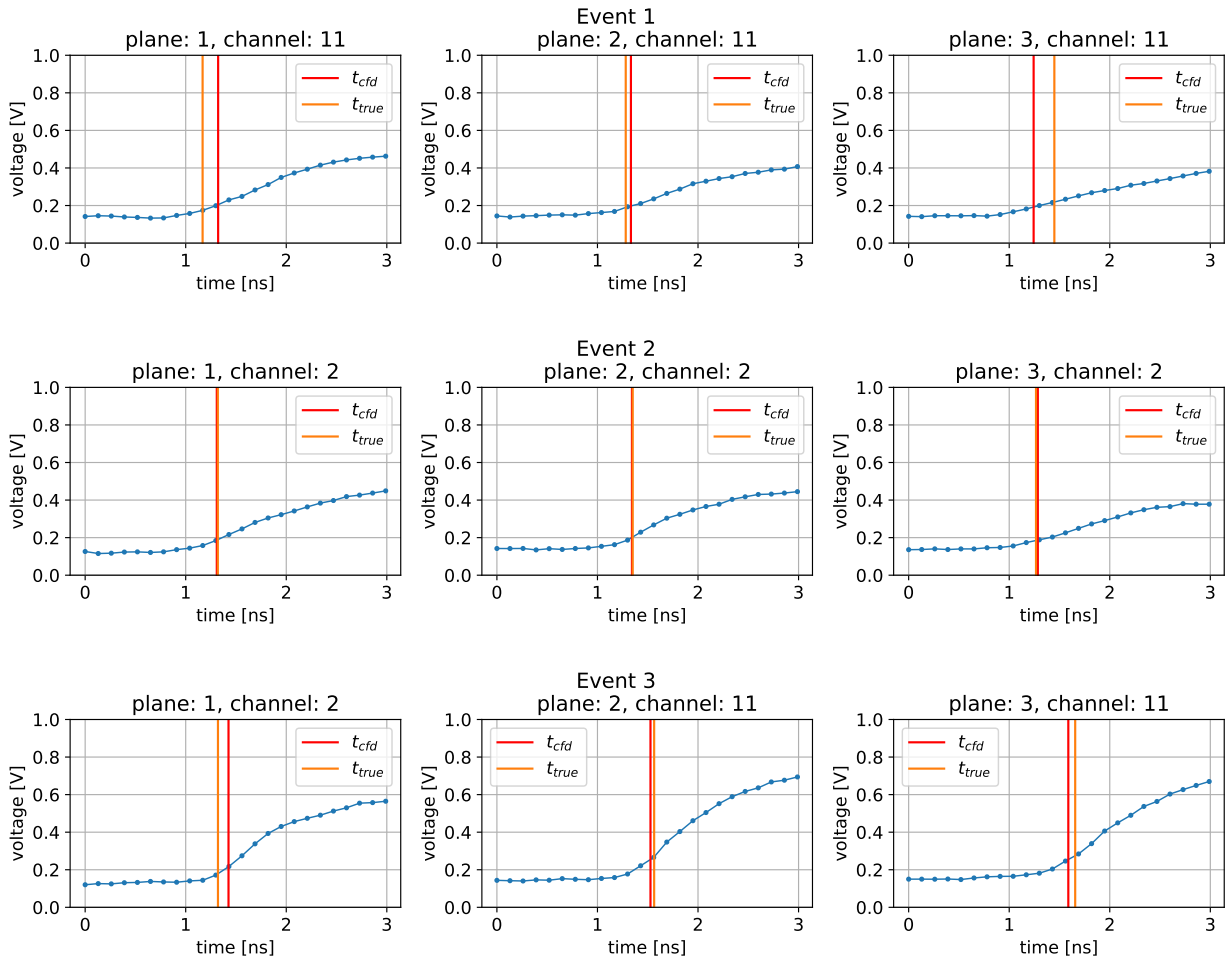


**Figure 4.14:** Example entries from the final dataset. All three waveforms are shown for each event. The event numbers are arbitrary. True (orange) and CFD (red) timestamps are marked. CFD and true timestamps are well aligned in some events, e.g. in event 2. Sometimes, however, one or more waveforms have some flaws and then the true and CFD timestamps can be shifted. For instance, in event 1, the waveform of plane 3 does not reach its maximum amplitude, therefore the CFD timestamp is shifted to the left. Unfortunately, this also impacts the true timestamps in other channels. Eliminating all faulty events like event 1 would be too time-consuming, so they are kept in the dataset. Event 3 is an example of an event in which one particle is measured by channels 2 and 11, as opposed to the majority of events in which a particle is detected by the same channel in each plane.

The numbers for respective channels in planes are not equal due to the fact, that the small fraction of events contains two hits in channel 2 and one in channel 11 (or vice versa). An example of such an event is shown in Figure 4.14 (bottom).

# 5. Exploration of the efficiency of selected neural network architectures

The chapter begins with a description of the common training characteristics and the approach used to tune the hyperparameters of neural networks. Next, all the tested architectures are described, along with their tunable hyperparameters and the corresponding optimal values. Finally, the optimal architectures are reported. The entire process was performed twice: once for data from all available channels and once for data only from channel (2, 11). It was assumed that the results for this channel could be generalised for other channels.

## 5.1. Hyperparameter tuning procedure

Neural networks are a highly configurable tool, which makes them adaptable to almost any type of problem. However, finding the optimal hyperparameter values is practically impossible. This procedure, called hyperparameter tuning, involves a trade-off between the quality of results and the time spent finding them. With easy access to powerful GPUs, hyperparameter tuning has become easier and often automated. The role of the expert is now limited to implementing the core of a network and setting viable hyperparameter ranges. There are numerous hyperparameter tuning frameworks available on the market to make the process even easier.

The networks in this thesis were implemented using the Keras interface of the TensorFlow library [50] for Python. The hyperparameter tuning framework used in this research is KerasTuner [51], one of the most commonly used tuning tools for TensorFlow. KerasTuner supports several optimisers, including grid search and random search algorithms. Grid search is the most basic algorithm, as it trains a network for every possible hyperparameter combination to find the best model. While it would be the best optimiser in the case of unlimited computing power, in practice, it takes too much time. This problem is solved with the random search algorithm, which trains a selected number of configurations with hyperparameter values chosen randomly. Here, time is not an issue, but it is unlikely to find the optimal model. To increase the likelihood, hyperparameter values can be chosen following a non-random strategy. KerasTuner implements the Bayesian optimisation strategy, which has been shown to achieve better results in a shorter amount of time compared to random search [52].

KerasTuner was used with the Bayesian optimiser to find the top five optimal models for each tested architecture. Instead of setting the same number of training epochs for each model, the early stopping strategy was employed. The maximum possible number of epochs was set to a large number so that every model could reach its loss minimum. Training a neural network is not deterministic, which means that the generalisation quality can be drastically different between two trained instances of the same model. To reduce the likelihood of selecting a model with high performance variation among its instances, the score of each model was calculated as the average loss value of two separately trained instances. The maximum number of tested hyperparameter configurations was set per architecture, considering the size of the hyperparameter space and training duration.

By default, KerasTuner does not support cross-validation, so each model was tested on the same 80%-20% training-validation split. To increase the probability of finding the optimal model, the

five-fold cross-validation procedure was run on the most promising hyperparameter sets returned by KerasTuner. As mentioned above, to further improve result precision, every model was trained twice for each fold, and the fold score was set to the average of these two executions.

The entire hyperparameter tuning procedure described above is summarised in Figure 5.1.

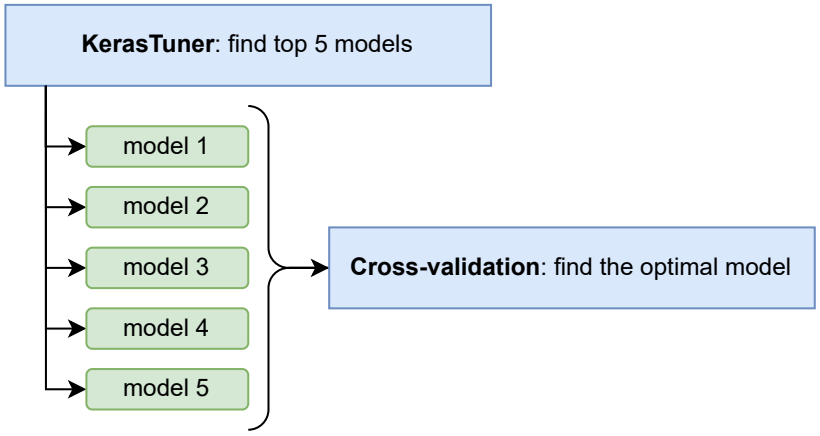


**Figure 5.1:** The two-step hyperparameter tuning procedure

## 5.2. Training characteristics and common remarks

The training setup was the same for every model during the hyperparameter tuning procedure. The mean squared error loss function was used together with the Adam optimiser with a learning rate of 0.01. Although the learning rate is sometimes included as a hyperparameter that requires tuning, in this case, there was no need to do so as the learning rate was reduced on a 10-epoch (one channel) or 5-epoch (all the channels) plateau, i.e. when there was no improvement in the training set loss value for the selected number of epochs. The reduction factor was 0.9. To account for different training times, the early stopping callback was used, which stopped the training when there was no improvement in the validation set loss value for 60 (one channel) or 30 (all channel) epochs. As the original loss function values were small, the loss weight was set to 1000, meaning that the loss values were always multiplied by 1000. Finally, the batch size was set to either 4096 (one channel) or 8192 (all the channels). The whole procedure was run on a GeForce RTX 3070 GPU. The tuning time varied depending on the architecture. The procedure took around an hour for MLP and reached up to 8 hours for UNet.

Preliminary experiments showed that the best results were obtained using ReLU activations. The most optimal layer order was:

1. dense/convolutional,
2. activation (ReLU),
3. optional batch normalisation,
4. optional dropout / spatial dropout.

It is usually suggested to place a batch normalisation layer between the dense or convolutional layer and the activation function [53]. However, in this particular case, this configuration resulted

in worse performance and a less stable training process, i.e. the network's results varied significantly among instances trained separately. As a result, it was decided to place an optional batch normalisation layer after the activation function. If used, a dropout layer was placed at the end, following the observations from [54].

Finally, neural networks typically benefit from normalising the input data. In this case, the input time series could be normalised in the same way as when running CFD (see Figure 4.4). However, preliminary experiments showed that introducing normalisation might even decrease performance. Therefore, normalisation was introduced as one of the hyperparameters. Adding a batch normalisation layer straight after the input was tested as another option as well.

## 5.3. MLP

The simplest neural network type, the multilayer perceptron (MLP), was the obvious choice for the first architecture. The network consisted of a group of stacked dense layers. It is commonly assumed that the importance of features increases as the network goes deeper. Since the number of important features (such as whole shapes and types of peaks) is smaller than the number of less important ones (such as less intuitive combinations of values from multiple samples), the number of neurons is often reduced in consecutive layers. This architecture used a hyperparameter to control this behaviour. Another hyperparameter determined the number of neurons in the penultimate dense layer, allowing the neuron counts in previous hidden layers to be inferred using these two hyperparameters.

Every dense layer except for the last one was followed by the most common activation function – ReLU. The last dense layer did not use any activation, because it had only one neuron and produced the final timestamp. Optionally, batch normalisation and dropout layers could be placed after ReLU activations using the order described in Chapter 2.

| Name | Description | Values |
|---|---|---|
| `n_hidden_layers` | Number of hidden layers | 1–8 |
| `units_mult` | The number of neurons in the last hidden layer was equal to 3 times `units_mul`. | 1, 2, 4, 8, 16, 32 |
| `unit_decrease_factor` | Multiplies the numbers of neurons in the previous layers, starting from the last hidden layer. | 1.0, 1.5, 2.0 |
| `bn` | Batch normalisation after every hidden layer | True/False |
| `input_bn` | Batch normalisation after the input layer | True/False |
| `normalize_signal` | Signal normalisation | True/False |
| `dropout` | Optional rate of dropout after every hidden layer | 0.0, 0.2, 0.5 |

**Table 5.1:** MLP: hyperparameters and their possible values

The hyperparameter options for the MLP network are described in Table 5.1. The promising hyperparameter sets returned by KerasTuner are reported in Table 5.2. The scores are the loss function values which is the mean squared error between the predicted and ground truth timestamp. The optimal MLP models are depicted in Figure 5.2. The results clearly show that it is better to place a

| Hyperparameter | M1 | M2 | M3 | MA1 | MA2 | MA3 |
|---|---|---|---|---|---|---|
| n_hidden_layers | 1 | 8 | 2 | 4 | 6 | 4 |
| units_mult | 32 | 8 | 32 | 32 | 32 | 16 |
| unit_decrease_factor | 1.5 | 1.0 | 1.0 | 1.0 | 2.0 | 2.0 |
| bn | False | False | False | False | True | False |
| input_bn | True | True | True | True | True | True |
| normalize_signal | False | False | False | False | False | False |
| dropout | 0.5 | 0.0 | 0.5 | 0.0 | 0.2 | 0.0 |
| **Parameter count** | 2593 | 4921 | 11905 | 30529 | 6389473 | 106849 |
| **Mean score** | 8.36 | 8.41 | 8.41 | 8.36 | 8.38 | 8.43 |
| **Score std** | 0.25 | 0.22 | 0.22 | 0.15 | 0.14 | 0.11 |

**Table 5.2:** MLP: top three promising hyperparameter sets returned by KerasTuner and their cross-validation scores for one channel (M1, M2, M3) and all the channels (MA1, MA2, MA3).

batch normalisation layer after the input instead of normalising the signals. The best model for one channel has a smaller number of parameters than the best model for all the channels. It is understandable, as the model for all the channels requires higher capacity due to the bigger diversity of the waveforms. What might be surprising is that regular batch normalisation after each layer is not used almost in any of the selected models.

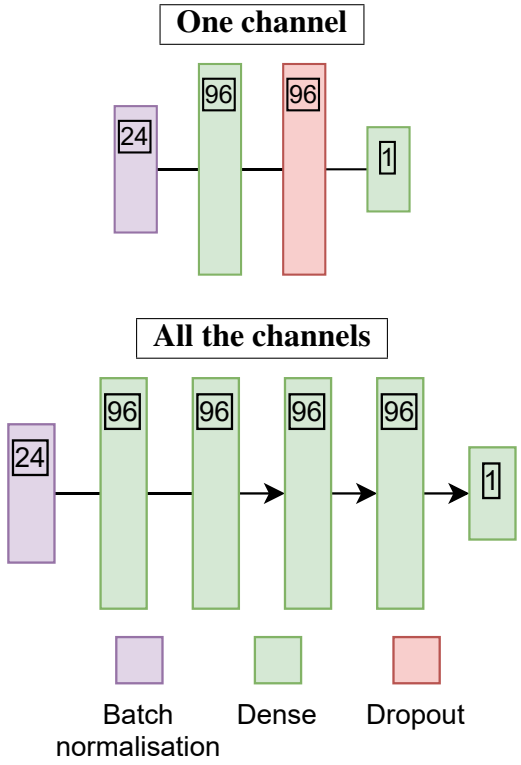


**Figure 5.2:** Optimal MLP models

## 5.4. Convolutional network

The next architecture was a convolutional neural network (CNN) based on VGG. Following recent trends, all convolutional layers had a small kernel size of 3. The network was divided into convolutional blocks, each consisting of one or more convolutional layers. Each convolutional layer could be followed by batch normalisation and/or spatial dropout layers. At the end of each block, max pooling was applied, which reduced the vector length by a factor of two. The number of convolutional filters remained constant within a block and was doubled in consecutive blocks.

The objective of these convolutional blocks was to extract features that were not time-related. These features were then processed in a single dense layer, which outputted the final timestamp. An MLP similar to the architecture described in the previous chapter could be placed between the convolutional part and the final dense layer. Its purpose was to improve the performance of extracting relevant information from the features produced by the convolutional blocks.

The hyperparameter options are described in Table 5.3. The loss function scores (MSE) are reported in Table 5.4. The optimal CNN models are depicted in Figure 5.3. Similarly to MLP, batch normalisation after input works better than signal normalisation. Surprisingly, networks for one channel and all the channels have similar numbers of parameters, although they are constructed differently. The one-channel networks rely on the convolutional layers more, while the networks for all the channels use shallow convolutional parts and deeper dense parts at the end.

| Name | Description | Values |
|---|---|---|
| `n_conv_blocks` | Number of convolutional blocks | 1-3 |
| `n_conv_layers` | Number of convolutional layers in a block | 1-3 |
| `conv_filters_mult` | The number of filters in the first block is equal to 8 times `conv_filters_mult`. | 1, 2, 4, 8 |
| `conv_spatial_dropout` | Optional rate of dropout after every convolutional layer | 0.0, 0.1, 0.2 |
| `n_mlp_hidden_layers` | Number of hidden layers in the MLP at the end of the network | 0-3 |
| `mlp_units_mult` | The number of neurons in the last hidden layer of the MLP is equal to 3 times `mlp_units_mul`. | 1, 2, 4, 8, 16 |
| `mlp_dropout` | Optional rate of dropout after every hidden layer of the MLP | 0.0, 0.2, 0.5 |
| `bn` | Batch normalisation after every hidden layer | True/False |
| `input_bn` | Batch normalisation after the input layer | True/False |
| `normalize_signal` | Signal normalisation | True/False |

**Table 5.3:** CNN: hyperparameters and their possible values

| Hyperparameter | M1 | M2 | M3 | MA1 | MA2 | MA3 |
|---|---|---|---|---|---|---|
| n_conv_blocks | 2 | 4 | 4 | 1 | 1 | 1 |
| n_conv_layers | 2 | 1 | 1 | 1 | 2 | 2 |
| conv_filters_mult | 2 | 4 | 2 | 4 | 2 | 4 |
| conv_spatial_dropout | 0.0 | 0.2 | 0.1 | 0.0 | 0.0 | 0.0 |
| n_mlp_hidden_layers | 2 | 1 | 1 | 2 | 3 | 3 |
| mlp_units_mult | 16 | 16 | 16 | 16 | 2 | 4 |
| mlp_dropout | 0.2 | 0.5 | 0.2 | 0.2 | 0.0 | 0.2 |
| bn | True | True | False | True | True | True |
| input_bn | True | True | True | True | True | True |
| normalize_signal | False | False | False | False | False | True |
| **Parameter count** | 64369 | 181153 | 57345 | 107873 | 14289 | 55873 |
| **Mean score** | 8.42 | 8.52 | 8.55 | 8.34 | 8.54 | 8.58 |
| **Score std** | 0.23 | 0.24 | 0.23 | 0.14 | 0.11 | 0.12 |

**Table 5.4:** CNN: top three promising hyperparameter sets returned by KerasTuner and their cross-validation scores for one channel (M1, M2, M3) and all the channels (MA1, MA2, MA3).

## 5.5. UNet

Using a convolutional network was a step forward compared to MLP, as it is better suited for time series processing. The next step was to use an architecture designed to identify key points, such as timestamps: UNet. The UNet-based network had a configurable depth, meaning the number of blocks in the encoder and decoder was a hyperparameter. The number of blocks could be set to 0, resulting in no encoder or decoder, just a bottleneck resembling a regular convolutional network. It is worth noting that the maximum depth was limited by the length of the time series, due to the max pooling layers which halve the time series length at each block.

Each block in the network contained a configurable number of convolutional layers. The number of convolutional filters was consistent within each block and was multiplied by two in subsequent levels. As before, each convolutional layer could be followed by batch normalisation and/or spatial dropout layers. In the decoder, the deconvolution blocks began with upsampling and dimensionality reduction using a convolutional layer with a kernel size of one. The outputs of these two operations were then concatenated with the corresponding skip layer from the encoder.

UNet required an additional preprocessing step: preparing the heatmaps. It is common to use a Gaussian centred on the key point, as in [55]. For this reason, each timestamp was processed with a Gaussian filter with a standard deviation of 1 sample. The loss function used was still MSE; however, the scores reported for UNet cannot be compared with those for the previous models, because of predicting heatmaps instead of just timestamps.

The available hyperparameter options are described in Table 5.5, and the loss function scores are reported in Table 5.6. The optimal UNet models are depicted in Figure 5.4. Similarly to the previous architectures, it is usually better to apply input batch normalisation instead of signal normalisation. Furthermore, that spatial dropout is employed in the optimal UNet models contrary to the convolu-

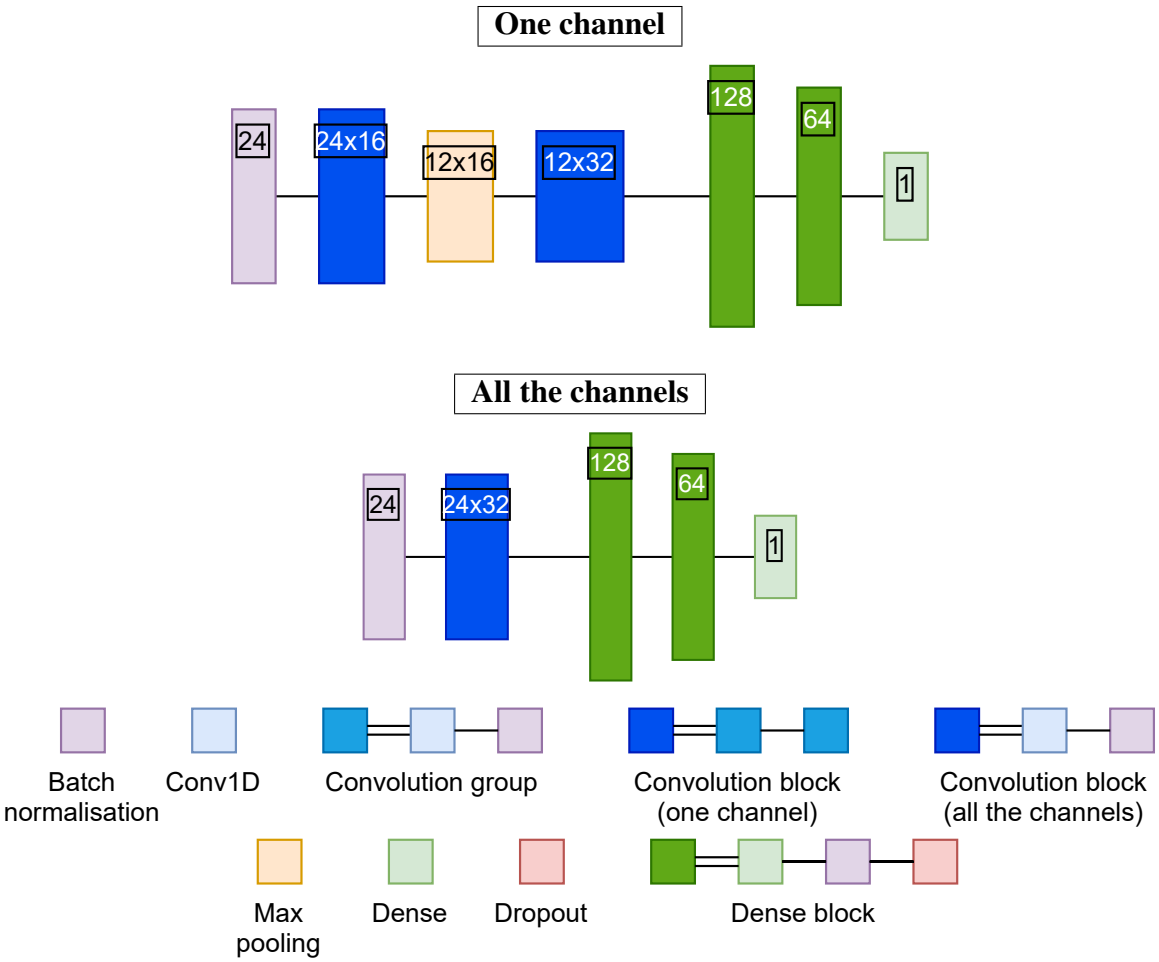


**Figure 5.3:** Optimal CNN models

| Name | Description | Values |
|---|---|---|
| `unet_depth` | Depth of the network in blocks | 0-3 |
| `n_conv_layers` | Number of convolutional layers in a block | 1-3 |
| `conv_filters_mult` | The number of filters in the topmost block is equal to 8 times `conv_filters_mult`. | 1, 2, 4, 8, 16 |
| `conv_spat_dropout` | Optional rate of dropout after every convolutional layer | 0.0, 0.1, 0.2 |
| `bn` | Batch normalisation after every hidden layer | True/False |
| `input_bn` | Batch normalisation after the input layer | True/False |
| `normalize_signal` | Signal normalisation | True/False |

**Table 5.5:** UNet: hyperparameters and their possible values

tional architectures where dropout was only used in the MLP at the end of the network. It is worth mentioning that, in almost every case, the deepest possible network in terms of the encoder and decoder size was used.

| Hyperparameter | M1 | M2 | M3 | MA1 | MA2 | MA3 |
|---|---|---|---|---|---|---|
| `unet_depth` | 3 | 3 | 2 | 3 | 3 | 2 |
| `n_conv_layers` | 1 | 1 | 2 | 3 | 3 | 2 |
| `conv_filters_mult` | 8 | 4 | 4 | 8 | 1 | 1 |
| `conv_spat_dropout` | 0.2 | 0.2 | 0.0 | 0.2 | 0.0 | 0.0 |
| `bn` | True | False | False | True | False | True |
| `input_bn` | True | True | False | True | False | True |
| `normalize_signal` | False | False | True | False | False | False |
| **Parameter count** | 1039969 | 259169 | 120641 | 3831201 | 46561 | 8433 |
| **Mean score** | 12.95 | 13.29 | 13.41 | 12.74 | 12.79 | 12.85 |
| **Score std** | 0.16 | 0.34 | 0.28 | 0.07 | 0.05 | 0.08 |

**Table 5.6:** UNet: top three promising hyperparameter sets returned by KerasTuner and their cross-validation scores for one channel (M1, M2, M3) and all the channels (MA1, MA2, MA3).

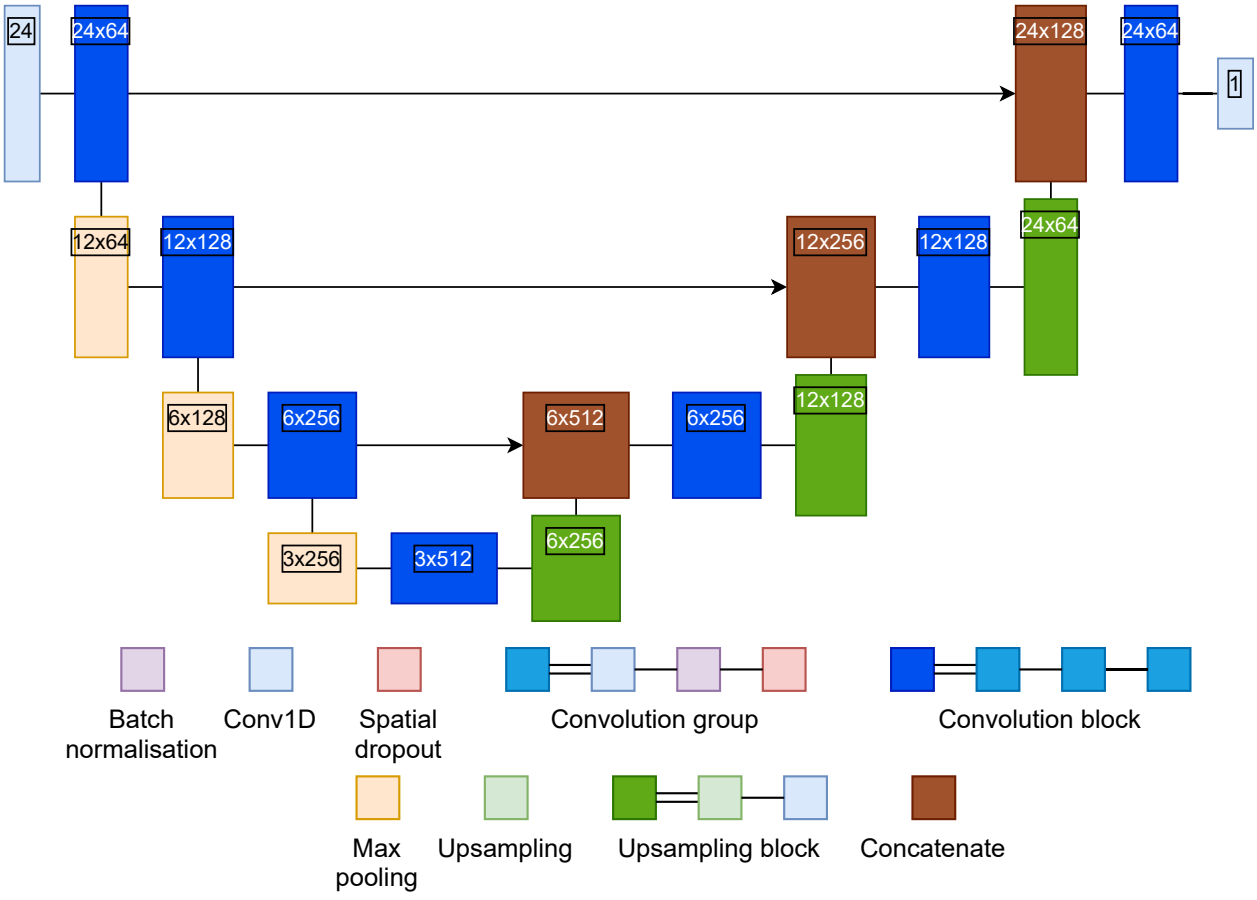


**Figure 5.4:** The optimal UNet model for all the channels. The model for one channel differs only by using one convolution group in the convolution block instead of three.

Another method of producing a heatmap for UNet was also tested. Following [47], each sample in the heatmap represented the distance to the timestamp. However, preliminary experiments showed

that this method was much less effective than using the Gaussian filter. Therefore, its results are not reported.

## 5.6. Recurrent network

The recurrent neural network (RNN) was the final architecture tested in this research. The main hyperparameter in this network determined whether to use LSTM or GRU layers. By default, there was only one layer, but it was possible to stack multiple layers together. The additional layers on top of the main one are referred to as hidden layers. Batch normalisation could only be added after input, as it is uncommon to use it after the recurrent layers. Dropout could not be configured for the same reason.

| Name | Description | Values |
|---|---|---|
| `rnn_type` | The recurrent layer used | LSTM, GRU |
| `n_neurons` | Number of neurons in each of the layers | 16, 32, 64, 128, 256, 512 |
| `n_hidden_layers` | Number of hidden recurrent layers | 0-1 |
| `input_bn` | Batch normalisation after the input layer | True/False |
| `normalize_signal` | Signal normalisation | True/False |

**Table 5.7:** RNN: hyperparameters and their possible values

| Hyperparameter | M1 | M2 | M3 | MA1 | MA2 | MA3 |
|---|---|---|---|---|---|---|
| `rnn_type` | GRU | LSTM | GRU | LSTM | GRU | GRU |
| `n_neurons` | 128 | 128 | 16 | 16 | 64 | 16 |
| `n_hidden_layers` | 1 | 1 | 0 | 1 | 1 | 0 |
| `input_bn` | True | True | True | True | True | True |
| `normalize_signal` | False | False | False | False | True | True |
| **Parameter count** | 149601 | 198369 | 1025 | 3377 | 37985 | 1025 |
| **Mean score** | 8.36 | 8.37 | 8.38 | 8.57 | 8.60 | 8.82 |
| **Score std** | 0.22 | 0.20 | 0.24 | 0.14 | 0.13 | 0.11 |

**Table 5.8:** RNN: top three promising hyperparameter sets returned by KerasTuner and their cross-validation scores for one channel (M1, M2, M3) and all the channels (MA1, MA2, MA3).

The hyperparameter options are described in Table 5.7. The RNN outputs a timestamp, hence MSE could be used as the loss function. The scores are reported in Table 5.8. The optimal RNN models are depicted in Figure 5.5. The cross-validation results show that models with one hidden recurrent layer achieved the best results. Adding a batch normalisation layer at the beginning of the network is helpful while normalising the signals usually does not provide any benefits. There is no clear performance distinction between using LSTM and GRU.

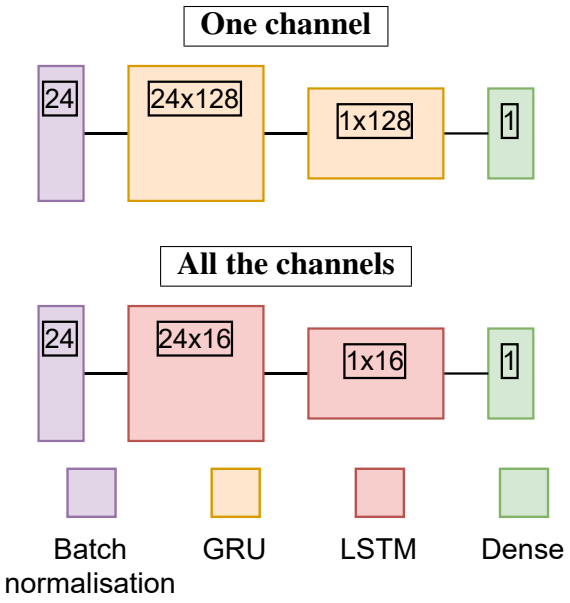


**Figure 5.5:** Optimal RNN models

## 5.7. The optimal models

The calculation of the loss function value for UNet was different from that of other models. In order to allow for comparison with other models, timestamps were obtained by fitting Gaussians to predicted heatmaps and extracting their means. After that, the UNet score could be calculated using MSE, just like other models.

However, a more advanced precision measurement method was chosen. A common approach is to create a histogram of the differences between the predicted and true timestamps and fit a Gaussian curve to it. This method aims to reduce the impact of outliers, or in other words, decrease the tail effect. The standard deviation of the Gaussian represents the precision. This thesis refers to this method as the 'difference histogram' method. An example is provided in Figure 6.1.

To find the optimal model, the cross-validation procedure was run again, separately for one channel and all the channels. However, this time only the best models per architecture were tested. To increase the reliability of the results, each fold's score was calculated as an average of the results of three independent instances. Finally, precision was measured using the difference histogram method. The results are presented in Table 5.9.

The cross-validation scores indicate that all the networks performed similarly. The relative difference between the best and worst precision is less than 2%, both in the case of one channel and all the channels. UNet has the best (smallest) precision in both cases, which is expected since UNet is best suited for annotating key points, such as timestamps. Therefore, UNet was chosen as the optimal network. It is worth noting that, based on the literature study, RNN was expected to achieve good precision compared to other network types due to its time series processing capabilities, but its performance was visibly worse than that of the UNet.

| One channel | | | | |
|---|---|---|---|---|
| **Model** | **MLP** | **CNN** | **UNet** | **RNN** |
| **Mean score [ps]** | 84.19 | 84.08 | 83.26 | 83.96 |
| **Score std [ps]** | 1.40 | 1.15 | 1.36 | 1.26 |
| **Parameter count** | 2593 | 64369 | 1212065 | 149601 |
| All the channels | | | | |
| **Model** | **MLP** | **CNN** | **UNet** | **RNN** |
| **Mean score [ps]** | 79.88 | 80.37 | 79.68 | 80.49 |
| **Score std [ps]** | 0.36 | 0.38 | 0.39 | 0.38 |
| **Parameter count** | 30529 | 107873 | 3831201 | 3377 |

**Table 5.9:** Cross-validation scores acquired using the difference histogram method for the optimal model of each architecture. The number of parameters is provided for each of the networks to compare their sizes.

# 6. Validation of the optimal model and further analysis

This chapter describes the validation of the optimal models identified in Chapter 5, using the dataset built in Chapter 4. The test part of the dataset is used for the first time to accurately assess the performance, following best practices of testing a machine learning model.

## 6.1. Comparison with CFD – one channel

Now that the optimal network for a single channel has been selected, it can be compared with CFD using the test dataset. Channel (2, 11) was used in the hyperparameter tuning procedure, and hence only data from this channel are used to assess the performance of the network. The difference histograms described in Chapter 5.7 generated both for CFD and the neural network are shown in Figure 6.1. The network improved the CFD's precision by 6.3%, from 90.4 ps to 84.7 ps. While the improvement may not seem significant, it is satisfying considering that CFD was used both to compute the ground truth values and as a reference for the networks.

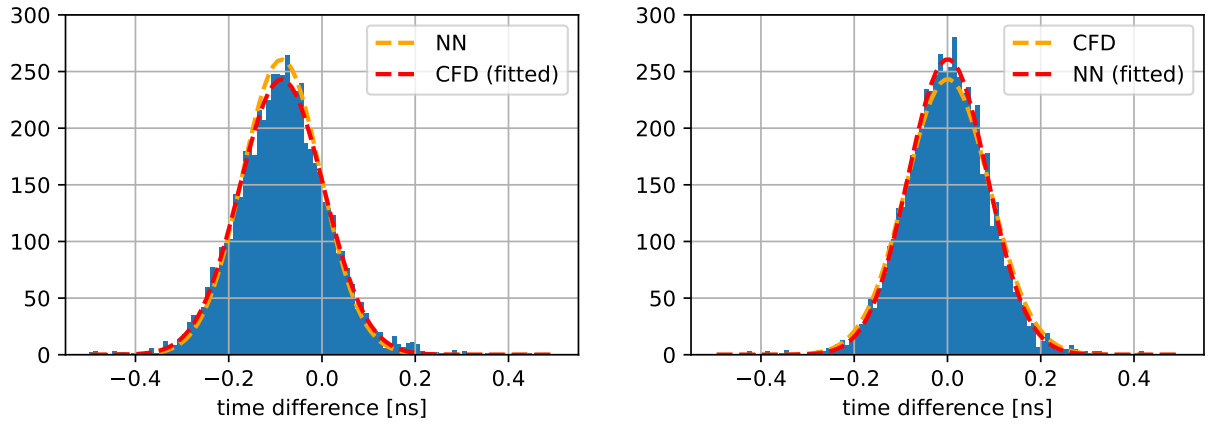


**Figure 6.1:** Difference histograms for CFD (left) and the optimal neural network (right) and the Gaussians fitted to the histograms. The network histogram and the corresponding Gaussian are visibly narrower, indicating a smaller error and better precision.

## 6.2. Comparison with CFD – all the channels

Although the hyperparameter tuning was not performed for each channel separately, it was assumed that the network chosen using the data from channel (2, 11) would work well for the other channels, too. To verify this, networks were trained separately using data from single channels and then tested on those channels. Additionally, they were tested on other channels to assess whether the network trained on one channel could also perform well on another.

The neural network tuned to be trained on the data from all the channels was trained using all available training data, i.e., training datasets from all the channels combined. It was then tested on each channel separately. Testing on all the channels at once could result in a group of smeared Gaussians instead of a single one due to potentially different mean values of difference histograms

for different channels. Therefore, it would be impossible to retrieve the true detector precision. Considering that the number of events in each channel is of the same order, the ultimate precision on the whole dataset can be calculated as the average of the results for single channels.

Additionally, another set of networks was trained using the optimal architecture for all the channels. However, instead of training on all the channels, each network was trained on a specific channel's dataset, which included all the channels except for this particular one used for testing. The purpose of these networks was to determine if any of the channels had significantly different data compared to the others.

The results of the experiments described above are presented in Table 6.1. As expected, networks trained on the same channels as the test channels showed the greatest improvements. However, testing a network on a channel different from the one used for training could result even in decreased performance compared to CFD.

| **Training channel** | **Test channel** | | | | | |
|---|---|---|---|---|---|---|
| | (1, 2) | (1, 11) | (2, 2) | (2, 11) | (3, 2) | (3, 11) |
| (1, 2) | **11%** | 2% | 4% | -2% | 5% | -11% |
| (1, 11) | 5% | **9%** | 8% | 3% | -5% | 9% |
| (2, 2) | 5% | 6% | 13% | 3% | 3% | 7% |
| (2, 11) | 6% | 3% | 6% | **6%** | 5% | 14% |
| (3, 2) | 6% | -22% | -9% | -5% | **12%** | -10% |
| (3, 11) | 5% | 2% | 4% | 4% | 0% | **18%** |
| All | 10% | **9%** | **15%** | 5% | 11% | 15% |
| Others | 8% | 7% | 13% | 4% | 6% | 12% |

**Table 6.1:** Precision improvements with respect to CFD obtained with many detector channels involved. The best improvement for each channel is highlighted.

The network trained on all the channels was expected to perform well for every channel, although not as well as networks trained on specific channels. This is because such a network would have to cover a wider spectrum of waveforms. Indeed, the network trained on all the channels showed promising results, with improvements close to the best ones. Moreover, the network showed improvements for every channel, proving that using more channels for training leads to higher generalisation capabilities.

Finally, the networks trained on all the other channels than the test one also showed good improvements, but slightly smaller than the network trained on all the channels. This means that while all the channels are slightly different from each other, no one is significantly outlying with respect to the others.

Although waveforms from different channels may appear similar, subtle differences can significantly impact network performance when working on a channel different from the one used for training. The similarity between specific channels can be validated by plotting waveforms on a 2D plane using dimensionality reduction methods, such as the simple Principal Component Analysis

(PCA) (dating back to 1901 [56]) or the more advanced t-Distributed Stochastic Neighbour Embedding (t-SNE) [57]. PCA works by projecting the data onto a lower-dimensional space, where each dimension is a linear combination of the original features, and the dimensions are ordered by the amount of variance they explain, which roughly corresponds to the amount of information they bring. In contrast, t-SNE is a nonlinear method that aims to preserve the local structure of the data in the low-dimensional space. Because of that, axes on t-SNE plots are not meaningful.

The purpose of this thesis is not to analyse different dimensionality reduction methods, so only the best visualisations are presented. T-SNE is used for the majority of the dataset visualisations because its plots are more compelling and potential clusters appear better separated. Figure 6.2 shows the 2D representation of waveforms from all the analysed channels generated using t-SNE. Due to the high computing requirements of t-SNE, only a subset of 1000 waveforms per channel was processed.

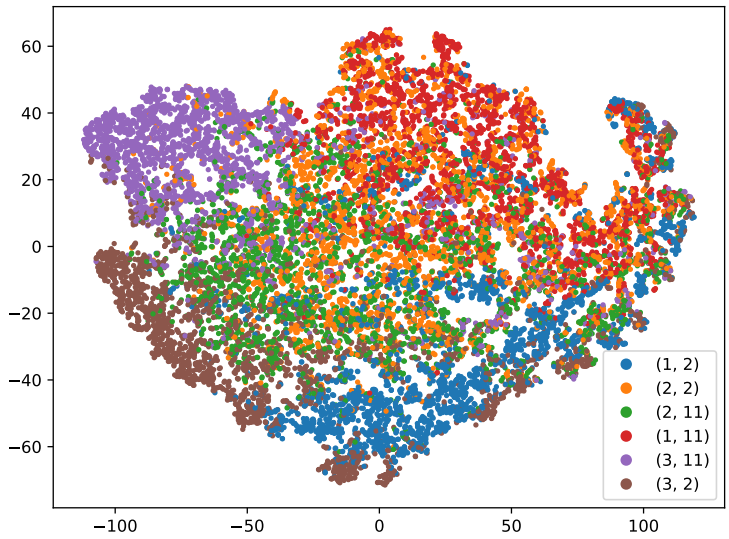


**Figure 6.2:** Waveforms from different channels plotted on a 2D plane using t-SNE. The order of plotting particular points is random.

The numbers from Table 6.1 can be easily correlated with the similarity between particular channels deducted from Figure 6.2. For instance, channels (3, 2) and (1, 11) are on the other sides of the plot meaning rather big differences between them. This is represented in the negative improvements when mixing these channels for training and testing. Conversely, the overlap between channels (2, 2) and (1, 11) seems to be high, and, as expected, the improvements for these channels are relatively high.

## 6.3. Pairwise precision and benefits for events with fewer hits

In the previous chapters, the reference timestamps were calculated using only two channels. This approach is useful when comparing the network with CFD. If the channel used to train the network was included in the calculation of the reference values, the comparison would be biased towards CFD as its timestamps would have a weight of one-third when computing the reference. Nonetheless, to train the network with the highest precision possible, three channels should be used to

compute the ground truth times of arrival. Unfortunately, the comparison with CFD using the same approach as previously is no longer sensible due to the bias. However, there is a way to assess the improvement.

Ideally, the time of arrival of a particle should be consistent across all the channels in which it was detected (also taking into consideration the $t_0$ shift explained in Chapter 4.1). On the other hand, large time differences between channel pairs mean that the detector precision is poor because they imply big uncertainties. Time differences can be visualised by plotting on a histogram. The precision between two channels, known as the pair precision (or coincidence precision), can be then determined by calculating the standard deviation of a Gaussian fitted to the histogram. The precision of a method (CFD or neural network) is finally expressed as the average of the pair precisions, and thus it is called the pairwise precision in this thesis. The result would benefit from weighing the average with the number of events considered for each channel. However, the number of events for each pair is of the same order, so the difference between the regular and weighted average is negligible.

It is important to notice that, if possible, it is still better to compute the precision using reference values – even if they are computed using CFD. The main advantage of having a real reference is the consistency of the results. Although CFD is much less robust than the neural network, it ensures that the timestamps are around the selected CFD threshold (even if the signal is not entirely in the measurement window and the true timestamps should be shifted). On the other hand, low pairwise precision only guarantees small differences between channels. For some events, this might mean predicting the time of arrival at the beginning of the rising edge, and for others – closer to the end.

While this problem has to be kept in mind, the networks in this thesis were trained exactly on the CFD averages and showed even smaller differences with respect to the average than the regular, one-channel CFD. Thus, pairwise precision can be safely used as the estimate of the precision improvement. Due to the reasons outlined in Chapter 4.1 it was decided to analyse the pairs within the same channels, i.e. (1, 2) vs (2, 2), (1, 11) vs (2, 11), etc. Figure 6.3 (top) presents the difference histograms and computed pair precisions for CFD. The pairwise precision is equal to 101.9 ps. The same histograms are also plotted for the neural network predictions in Figure 6.3 (bottom). The pairwise precision is equal to 86.7 ps, giving an improvement of around 15%, which is in line with the improvements from previous chapters.

The pair precisions can be deconvolved, and the estimated precisions of each sensor can be extracted using the following equations:

$$\begin{aligned}\sigma(1,2)^2 &= \sigma(1)^2 + \sigma(2)^2\\ \sigma(1,3)^2 &= \sigma(1)^2 + \sigma(3)^2\\ \sigma(2,3)^2 &= \sigma(2)^2 + \sigma(3)^2\end{aligned}$$

where $\sigma(1,2)$ stands for the pair precision between planes 1 and 2 (for the same channel: 2 or 11), and $\sigma(1)$ is the deconvolved precision of channel 1. These equations can be easily transformed

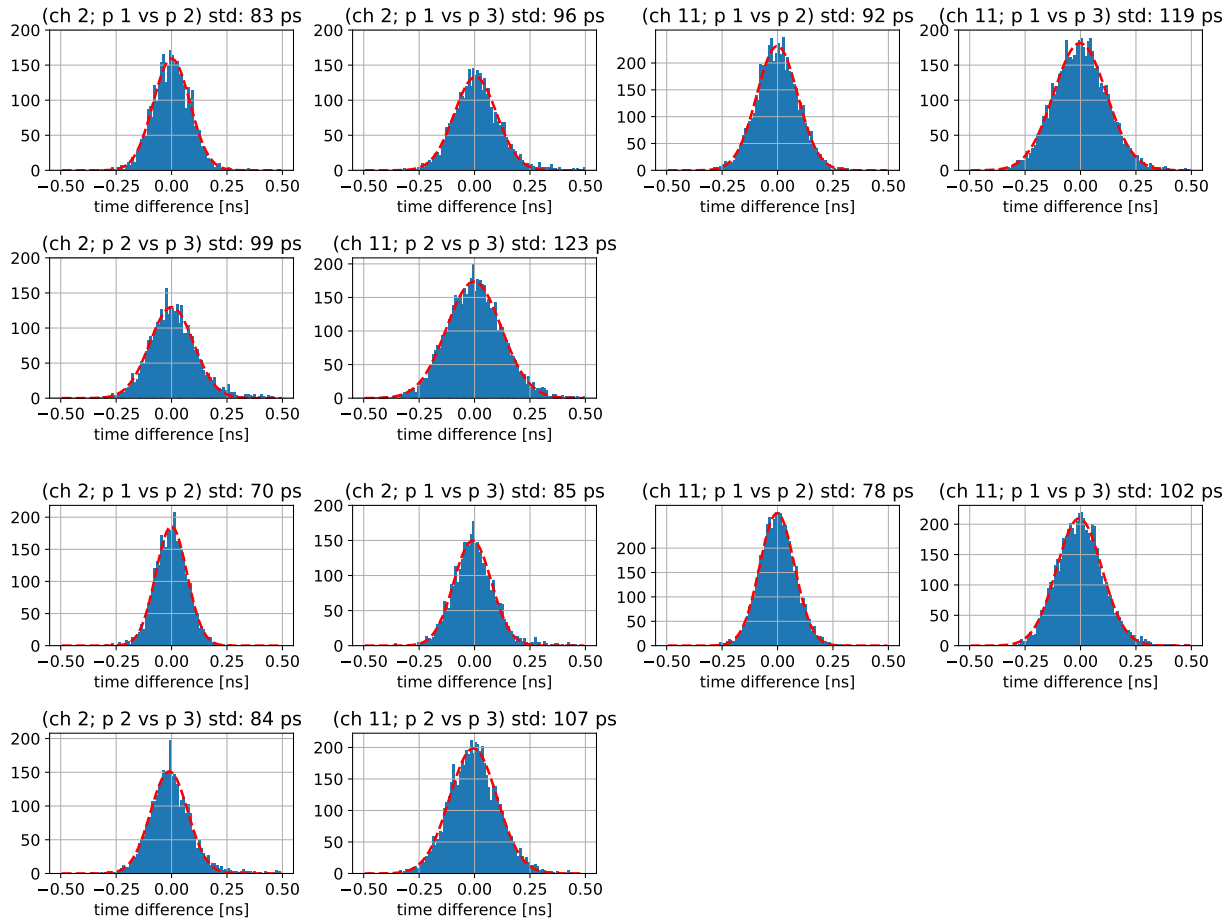


**Figure 6.3:** Pair difference histograms for CFD (top) and the neural network (bottom) together with pair precisions computed as the standard deviations of the Gaussians fitted to the histograms.

to:

$$\sigma(1) = \sqrt{(\sigma(1,2)^2 + \sigma(1,3)^2 - \sigma(2,3)^2)/2}$$
$$\sigma(2) = \sqrt{(\sigma(1,2)^2 + \sigma(2,3)^2 - \sigma(1,3)^2)/2}$$
$$\sigma(3) = \sqrt{(\sigma(1,3)^2 + \sigma(2,3)^2 - \sigma(1,2)^2)/2}$$

The deconvolved precisions together with the improvements with respect to CFD are listed in Table 6.2. As previously, the improvements are not spectacular due to the quality of the dataset. However, even the smallest improvement of 11% is certainly noticeable.

Since the network was trained to predict the average of the CFD timestamps from three selected channels, it cannot work better than its reference. Thus, the benefit of using the network for such events is questionable, as it is easier to simply use the CFD average. However, one of the advantages of this solution is that it is expected to have the same performance even for events with fewer hits than three available. As shown in Figure 4.3, the number of events with only one or two hits is noticeable. The benefit for the events with only one hit is obvious – the CFD average would take

| Channel | CFD [ps] | NN [ps] | Improvement |
|---|---|---|---|
| **(1, 2)** | 56.3 | 50.1 | 11% |
| **(2, 2)** | 60.7 | 48.2 | 21% |
| **(3, 2)** | 77.6 | 67.0 | 14% |
| **(1, 11)** | 61.2 | 50.2 | 18% |
| **(2, 11)** | 69.2 | 59.6 | 14% |
| **(3, 11)** | 101.7 | 87.4 | 14% |

**Table 6.2:** Deconvolved precisions for CFD and the neural network (NN) and improvements of the network with respect to CFD

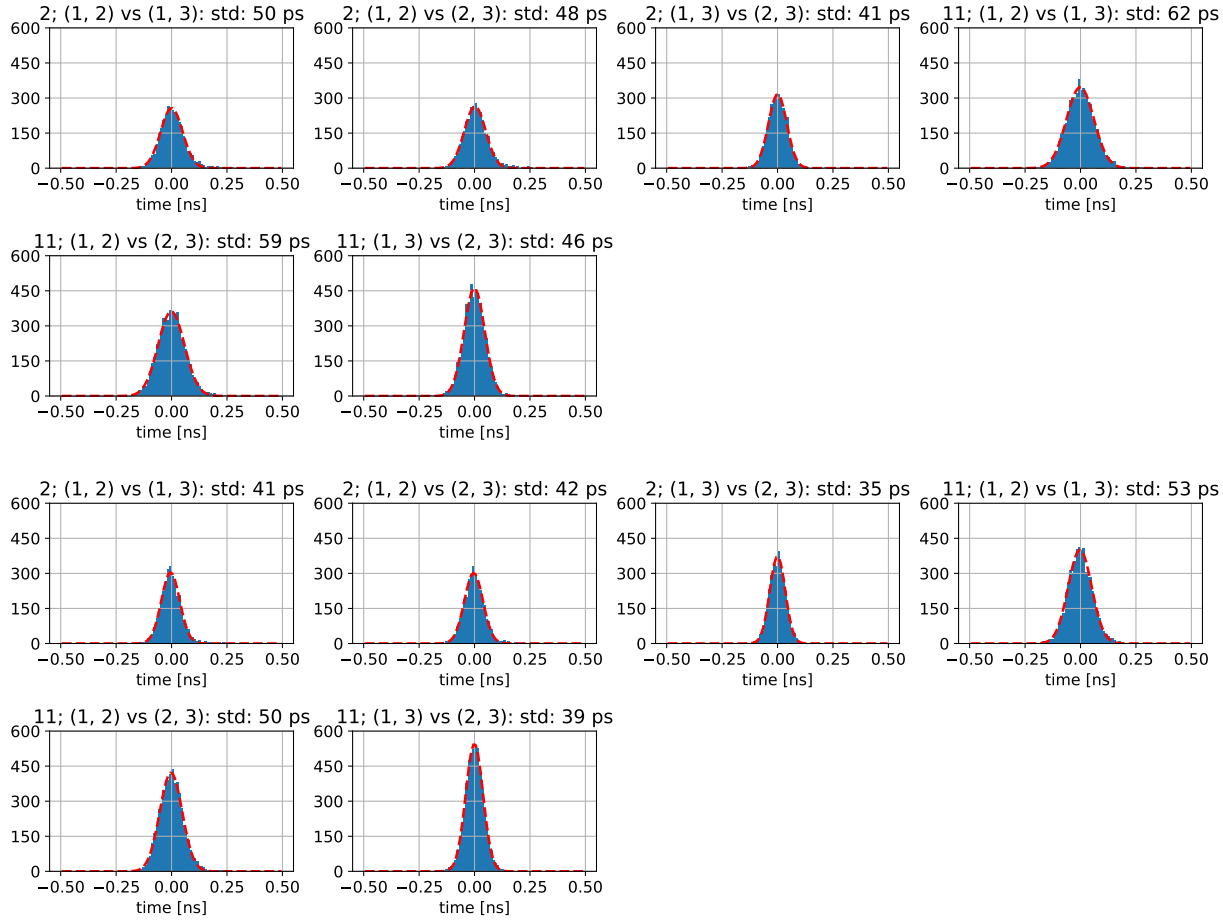


**Figure 6.4:** Two-channel pair difference histograms for CFD (top) and the neural network (bottom) together with the precisions computed as the standard deviations of the Gaussians fitted to the histograms. The title of each histogram is formatted as follows: "channel; (plane pair 1) vs (plane pair 2)"

only one number into account. For the two hits, the average of two CFD timestamps might be more precise than the neural network prediction from a single channel. However, the network average is expected to be even more precise than the CFD average. This is verified by building the difference histograms of the two-channel averages (the average of planes 1 and 2 versus the average of planes

1 and 3, etc.) which are shown in Figure 6.4 together with the two-channel pair precisions. As expected, the network is better than CFD with improvements ranging from 13% to 17%.

## 6.4. Tests on noisy and saturated data

One of the advantages of the neural network is that it can learn anything and is not limited by the shape of the signal as CFD. This feature is particularly useful when predicting the time of arrival for noisy or saturated waveforms (examples in Figure 4.6). The gain of the neural networks is expected to be high due to the fact that CFD cannot be used correctly.

Computing the reference data was slightly more difficult in this case. Only events in which one of the waveforms was either noisy or saturated could be included when building the dataset. Otherwise, the quality of the reference timestamps would decrease notably. As a result, the dataset is smaller compared to the original one, with the number of events per channel ranging from 300 to 1100. This size may not be sufficient to train the network without significantly biasing it towards the training samples. Therefore, the data were merged, and the network was trained only on all the channels. Additionally, due to the small volume, the precisions were computed as the standard deviations of the differences without plotting the difference histograms, which might be too ragged to apply a Gaussian fit.

Table 6.3 compares the improvements of the neural network to CFD in two variants. The first one involved training the network only with noisy or saturated data. In the other one, noisy and saturated data were appended to the regular dataset. Although the improvements for the second variant are high, they are much worse than for the first one. The main reason for this is that the task is less difficult for the network in the second variant, as the data similar to the test dataset (noisy and saturated) constitute only a small chunk of the training dataset. Another relevant aspect is that even though the data were filtered to contain only entries in which one waveform is either noisy or saturated, the rest of the waveforms might be on the edge of the amplitude cuts (see Figure 4.7), decreasing the quality of the ground truth timestamps.

| **Channel** | **(1, 2)** | **(1, 11)** | **(2, 2)** | **(2, 11)** | **(3, 2)** | **(3, 11)** |
|---|---|---|---|---|---|---|
| **Improvement v. 1** | 40 % | 34 % | 34 % | 23 % | 20 % | 54% |
| **Improvement v. 2** | 20 % | 31 % | 23 % | 20 % | 7 % | 39% |

**Table 6.3:** Improvements of the neural network with respect to CFD for the dataset consisting of noisy and saturated events. Variant 1: trained using the whole dataset with appended noisy and saturated records, variant 2: trained only using noisy and saturated entries.

## 6.5. Tests on data from the other LHC sector

The network can be validated using the data from another sector to check the difference between the waveforms in the two sectors. There is no point in a network trained using a particular channel from the RP in sector 56 as the detector in sector 45 is a different device. Even though the same channel numbering is used for the RP in sector 45 (plane 0, 1, 2, 3; channel 2, 11), these are different planes with different specifics. Hence only the network trained on all the channels was used.

The dataset for sector 45 was prepared similarly to the main dataset, but with a few configuration differences, such as different maximum amplitude histogram cuts. However, in the case of sector 45, plane 2 was off. Only channels (0, 2) and (3, 2) looked reasonable on the maximum amplitude histograms for the remaining channels. To build a dataset similar to the base one, three channels were needed, so channel (1, 2) was also included, even though it reduced the overall number of events to only 4700, compared to around 16000 (ch. 2) or around 28000 (ch. 11) for sector 56. The cuts are shown in Figure 6.5. Table 6.4 lists the improvements in comparison to CFD. Additionally, another network was trained on the combined data from sector 45 for comparison purposes.

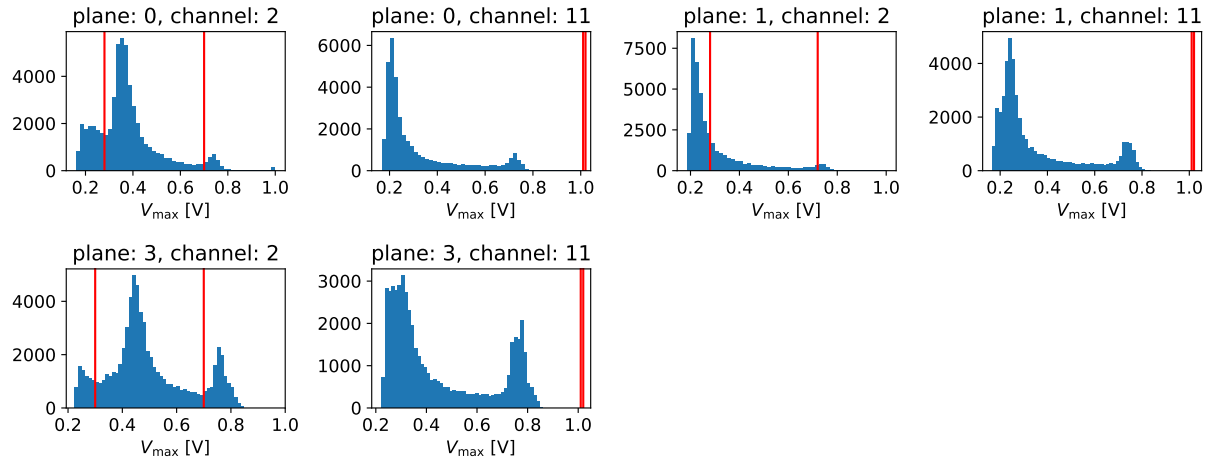


**Figure 6.5:** Max voltage (amplitude) histograms for each readout channel in the RP in sector 45. Only events between the red lines are kept for further processing. Plane 2 is not present since it was not working in sector 45. Only channels (0, 2) and (3, 2) can be properly filtered from noisy and saturated events. However, to retain the dataset consistency and compute the reference values using two timestamps, channel (1, 2) is also included which significantly reduces the number of analysable events.

| **Channel** | **(0, 2)** | **(1, 2)** | **(3, 2)** |
|---|---|---|---|
| **Improvement (default)** | 2% | -5 % | 3% |
| **Improvement (trained on s. 45)** | 7 % | 13 % | 5% |

**Table 6.4:** Improvements of the network trained on all the channels in the RP in sector 56 ("default") and on all the channels in the RP in sector 45 ("trained on s. 45") with respect to CFD on the channels available in the RP in sector 45.

As expected, the network trained with data from sector 56 is not robust enough to work for sector 45. The waveforms from sector 45 are not similar enough to those from sector 56, so the network makes predictions based on input that is notably different from the waveforms it was trained on. The plot of waveforms reduced to two dimensions by t-SNE in Figure 6.6 shows that a substantial portion of the data from sector 45 is significantly different from any waveform in sector 56 (presumably channel (1, 2), which is the only one with negative improvement). Consequently, the network trained on data from sector 45 achieves much better improvements.

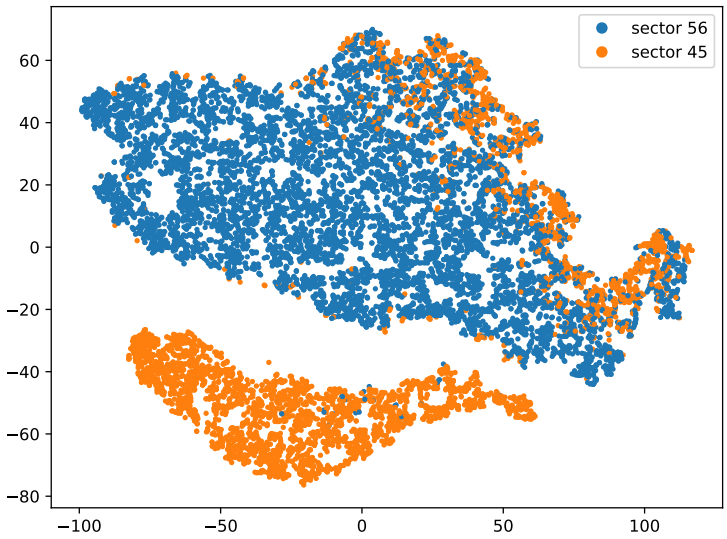


**Figure 6.6:** Waveforms from both sectors plotted on a 2D plane using t-SNE. The order of plotting particular points is random.

## 6.6. Tests on data from other LHC runs

The next step is to test the network on the data from another run. So far, only one LHC run was used – run 355207. The main reason to select this particular one was the fact that this was one of the few runs containing SAMPIC readouts. Moreover, the calibration object needed to properly build the waveforms in the CMSSW software framework was already prepared and easily accessible.

Other runs containing SAMPIC data can be used for validation, but they require their own calibration objects. These objects were not available for runs other than the main run 355207, so they had to be prepared for this thesis. The calibration process is not the essence of this thesis, hence it is not described here. It is performed in CMSSW and does not involve ML methods used in this research. More details on the SAMPIC calibration can be found in [58].

Run 354322 was selected as the second run to be tested in this research, as it was taken on 23 June 2022, around the same time as Run 355207, which was taken on 7 July 2022. Additionally, the RP configuration was the same for both runs, with the same sensors mounted in both sectors. Thus, run 354322 is a suitable choice for testing the generalisation capabilities of the network. The data for the RP in sector 56 in run 354332 turned out to be similar to the data from the main run. Moreover, even plane 0 could be analysed, as it recorded a significant number of good waveforms, unlike run 355207. However, due to the lack of a clearly visible pattern of noise, good data, and saturation, it is not used. The maximum amplitude histograms, together with the cuts, are shown in Figure 6.7. The rest of the processing was similar to the main run. The final dataset for run 354322 was composed of around 10,000 events per each channel.

The networks trained using the data from run 355207 can be used to validate their performance on data from run 354332. For comparison purposes, another set of networks was trained on run 354332 data to check if training on the same run that is used for testing increases the precision significantly. Moreover, other instances of the same network architecture were also trained on the

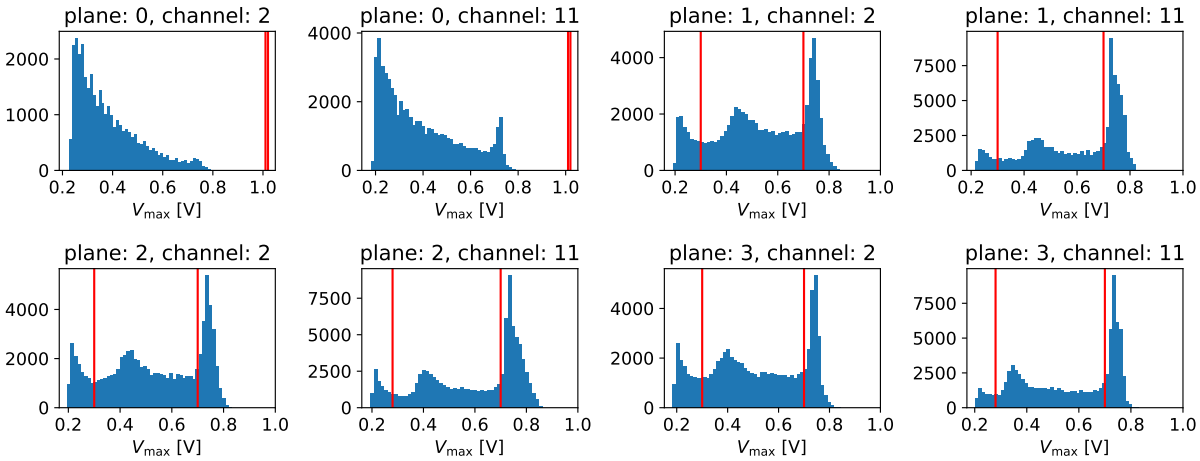


**Figure 6.7:** Max voltage (amplitude) histograms for each readout channel in sector 56 for run 354322. Only events between the red lines are kept for further processing. Due to lack of the pattern of noise, good data and saturation for plane 0, filters are placed on values bigger than 1 to filter out all the events.

predecessor of run 354332 – run 354331 which was taken at the same date as run 354332 and was a part of the same fill (7856). It can be expected that the waveforms of run 354331 are more similar to run 354332 than the waveforms of run 355207 which was taken after about two weeks. This time, the relation between runs is better visible when using PCA instead of t-SNE. The plot in reduced dimensionality is shown in Figure 6.8. Although, at first glance, all the runs seem to have very similar waveform distribution, run 355207 has less density of waveforms on the right side of the PCA plot. Thus, the network trained of run 355207 is expected to achieve worse results for those waveforms from run 354332 which are mapped to the right sector of the plot.

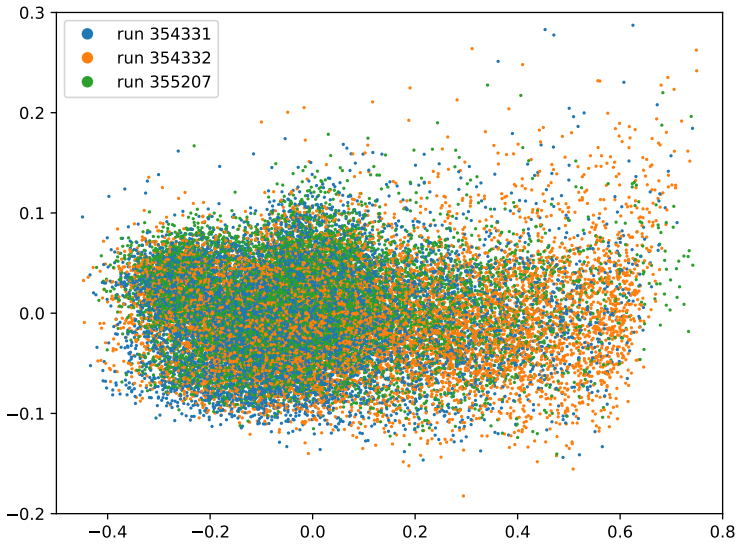


**Figure 6.8:** Waveforms from the LHC runs 354331, 354332 and 355207 plotted on a 2D plane using t-SNE. The order of plotting particular points is random.

Finally, a set of networks was trained also on the data from one run from a bunch of 2022 fills containing SAMPIC. Fill 7856 was not included in the dataset for these networks as this fill included run 354332 used for testing. The following runs were used: 355207 (the default one for the majority of experiments), 355445, 355558, 355680 and 355769. The last one – run 355769 – took place on 18 July 2022 which means that only a small period of LHC data-taking is used as the earliest run used – run 354331 – was recorded on 23 June 2022.

The results of the network sets described in this chapter, which were trained on all the channels and on single channels, are listed in Table 6.5.

| **Trained on** | **Channel (run 354332)** | | | | | |
|---|---|---|---|---|---|---|
| | (1, 2) | (1, 11) | (2, 2) | (2, 11) | (3, 2) | (3, 11) |
| Run 354332 (all the channels) | **14** % | 8 % | 13 % | 7 % | 18 % | 21% |
| Run 354332 (single channels) | 13 % | **9** % | **14** % | 8 % | 19 % | **20**% |
| Run 354331 (all the channels) | 13 % | **9** % | 13 % | 7 % | 15 % | 18% |
| Run 354331 (single channels) | 13 % | **9** % | 14 % | 8 % | 19 % | **20**% |
| Run 355207 (all the channels) | 9 % | 8 % | 13 % | 5 % | 12 % | 12% |
| Run 355207 (single channels) | 9 % | 8 % | 12 % | **9** % | 16 % | 19% |
| Selected fills from 2022 (all the channels) | 10 % | 8 % | **14** % | 7 % | 18 % | 1% |
| Selected fills from 2022 (single channels) | 9 % | **9** % | 12 % | 7 % | **20** % | 3% |

**Table 6.5:** Improvements of the neural network with respect to CFD for the data from run 354332. The best improvements per each channel are marked in bold. The networks used in "Selected fills from 2022" do not include fill 7856, since run 354332 is a part of this fill.

As anticipated, the results for run 355207 are notably worse, whereas the results for runs 354331 and 354332 are very similar, or even the same, with an uncertainty of 1-2 percent of improvement. This was expected, as these runs come from the same fill. Typically, the conditions in the LHC do not change much during a fill.

The results for the networks trained on other fills than the one including run 354332 are surprising. In general, except for two channels, they are better than the networks trained on run 355207. In the case of channel (3, 2), they are even slightly better than the network trained on the test run. The weirdest results were achieved for channel (3, 11), as the networks trained on many fills achieved very small improvements compared to the networks trained on other datasets. A possible cause of this effect might be worse waveform, and in turn, reference time quality. This could explain the poor quality of the results, as the network cannot perform well given inconsistent reference data. Another possibility would be a significant difference in the shapes of waveforms of channel (3, 11) among various runs. However, Figure 6.9 shows that the waveforms from run 354332 overlap with other runs, meaning that there is no significant difference.

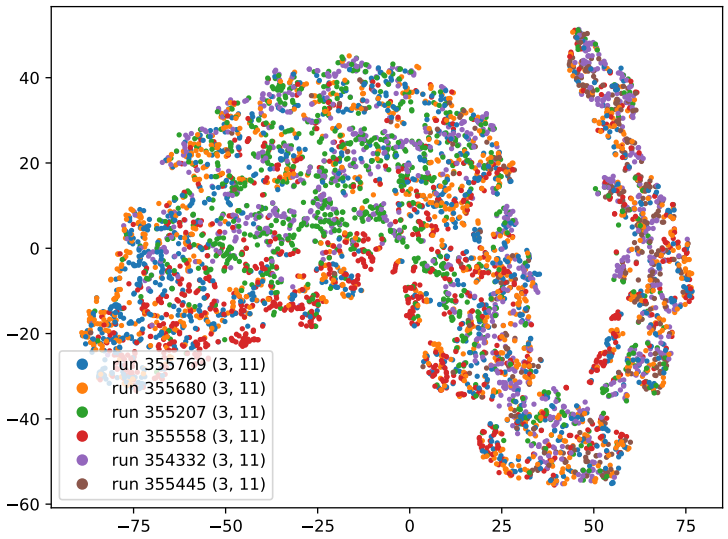


**Figure 6.9:** Waveforms of channel (3, 11) from the 2022 LHC runs used in Chapter 6.6 plotted on a 2D plane using t-SNE. The order of plotting particular points is random.

## 6.7. Test on 2023 data

The final test was to determine how well a network trained on data from one year could handle data from another year. PPS did not collect much SAMPIC data in 2023, but one run (run 370138) is available. Although the detector underwent many changes between the SAMPIC collection periods in 2022 and 2023, the same sensors were mounted in both sectors. Additionally, two more RPs were mounted, one in each sector, providing a total of two RPs per sector. However, due to the lack of data for these RPs in 2022, they are not used in this research. For this test, only data from one RP is used, namely the RP in sector 56 from the cylindrical station, which is the same RP as the one used in the main numerical experiment of this thesis.

As shown in the amplitude histograms presented in Figure 6.10, the waveform quality significantly decreased. This is mainly due to a different event-triggering logic used for the LHC data. In 2022, a special trigger was employed at the CMS level to block unwanted events. However, this trigger was not used for the 2023 run with available SAMPIC data, resulting in an increased amount of noise. Additionally, the quality of sensors gradually deteriorates over time, which may have contributed to the decrease in quality. To address this issue, only events with at least three hits in three different planes were utilised.

The waveform characteristics were analysed using a T-SNE mapping for two selected runs from 2022 and the 2023 run in Figure 6.11. As expected, the events from 2023 are much different to the ones from the 2022 runs. Therefore, the network performance was not expected to be high. A set of networks was trained on the data from one run per fill with available SAMPIC data from 2022. The results are shown in Table 6.6. As expected, the results are very poor, oftentimes with large negative numbers.

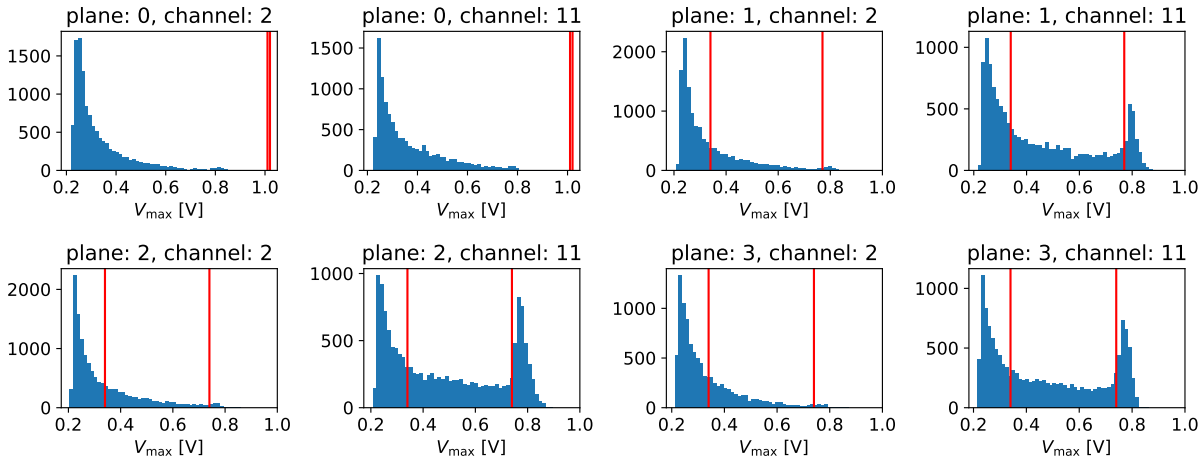


**Figure 6.10:** Max voltage (amplitude) histograms for each readout channel in sector 56 for run 370138. Only events between the red lines are kept for further processing. Following the way the 2022 datasets were built, no data are selected for both channels of plane 0.

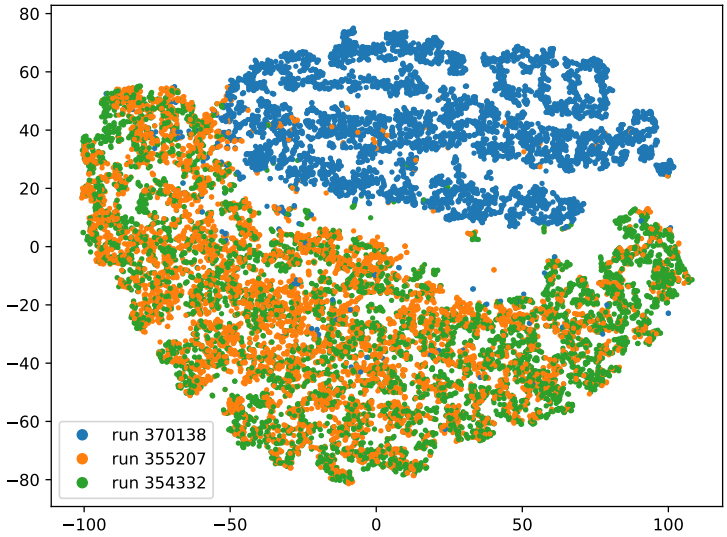


**Figure 6.11:** Waveforms from two selected 2022 runs (354332 and 355207) and the 2023 run (370138) plotted on a 2D plane using t-SNE. The order of plotting particular points is random.

| **Trained on** | **Channel (run 370138)** | | | | | |
|---|---|---|---|---|---|---|
| | (1, 2) | (1, 11) | (2, 2) | (2, 11) | (3, 2) | (3, 11) |
| Selected fills from 2022 (all the channels) | 4 % | -59 % | 13 % | -29 % | 14 % | -21% |
| Selected fills from 2022 (single channels) | -106 % | -19 % | 17 % | -11 % | -2 % | -5% |

**Table 6.6:** Improvements of the neural network with respect to CFD for the data from run 370138

# 7. Conclusion

The purpose of this chapter is to provide a summary and conclusion of the entire thesis, along with commentary on the conducted experiments and achievements. Additionally, potential goals for future research are outlined.

## 7.1. Summary

Chapter 1 contains an introduction to the research and defines numerous essential concepts. The thesis goal is also stated in this chapter. This thesis heavily relies on many concepts related to deep learning. To avoid explaining them in various places, brief descriptions are provided in Chapter 2. The last chapter not focusing on the actual research carried out for this thesis is Chapter 3. It explains the current state-of-the-art for both classical methods like CFD and deep learning approaches.

Data selection and preprocessing, as well as the creation of the dataset used to train and validate neural networks, were crucial in this thesis. Therefore, Chapter 4 is devoted to describing this process. The main dataset consisted of events from a single LHC run from 2022, and only data from the RP in sector 56 were used. However, other datasets built for secondary experiments employed sector 45 and other runs from both 2022 and 2023. Since the SAMPIC readout device was connected to two channels per detector plane, only eight channels could be analysed for an RP. Furthermore, after preliminary analysis, it was decided not to use data from plane 0 (both channels) due to the worse quality of the waveforms and higher noise ratio. Although the other channels contained some noisy or saturated events, they were filtered out by plotting histograms of amplitudes and applying minimum and maximum amplitude cuts. This was the most important preprocessing stage. After going through other, less important preprocessing stages, Chapter 4 explains the process of constructing the dataset, which includes waveforms and their corresponding reference ground truth time of arrival timestamps. The true timestamps for specific channels were obtained by computing the average of CFD timestamps from the other two channels.

Chapter 5 focuses on selecting the optimal model for both a single detector channel and all the channels at once. The chapter describes the hyperparameter tuning procedure that uses KerasTuner to identify the top five most viable networks and then selects the optimal one through cross-validation. The tuning procedure was performed on four network architectures: multilayer perception, regular convolutional network, UNet-based convolutional network, and recurrent neural network. The goal was to find the optimal values of hyperparameters defined in this chapter. UNet was found to be superior in both cases – for a single channel and for all the channels. However, the differences in results between the network architectures were not significant.

Finally, Chapter 6 focuses on using the optimised networks and analysing the results. First, the network trained on a single channel was tested on that particular channel, resulting in a 6% improvement with respect to CFD. Then, the networks were trained on each channel separately and on all the channels together. The best improvements were achieved by training the network only on the channel used for testing (6% to 18%, depending on the channel). However, testing on another channel than the one used for training resulted in much worse improvements, often negative (-22% to 14%), meaning worse performance than CFD. The network trained on all the channels showed

good improvement for each channel (5% to 15%), but in general, it was slightly worse compared to using one network per channel. The next experiment focused on analysing pairwise precisions, with results similar to the previous tests and an improvement achieved for each channel. Further experiments focused on analysing different datasets. First, a dataset with noisy and saturated waveforms was used. Afterwards, another sector was tested, followed by other runs from 2022 and 2023.

## 7.2. Discussion

First of all, this thesis is a follow-up to previous research that used data from a test beam in Germany, where PPS timing sensors were installed in 2020 [14]. The goal of that research was to test the deep learning approach and compare it with CFD, which aligns with the goal of this thesis. The author of this thesis is a co-author of an article on using neural networks on the test beam data, which has been pre-accepted for publication in the Computer Science journal following the author's presentation at the KUKDM 2023 conference in Zakopane, Poland[8]. A similar talk was also given at the FAST 2023 conference on Elba Island in Italy[9]. The research showed improvements of around 20% using the UNet-based network. The main difference was the use of another, much more precise detector: the MCP-PMT (MicroChannel Plate Photomultiplier Tube), together with the diamond sensors. The MCP-PMT was used to obtain the reference timestamps used to train the networks. Unfortunately, there is no more precise timing PPS detector in the LHC, so another way had to be found to build the reference dataset for this thesis. It was decided to use an average of other channels, which is more precise than just a CFD timestamp from one channel, but still limited by the precision of CFD. This limitation must be kept in mind while analysing the results. The network can only be as good as its reference, and in this case, the reference precision was very poor.

Despite the bad reference timestamps, the network was able to improve the CFD performance. Moreover, these improvements were observed even with the pairwise precision metric, which does not depend on CFD and reference quality, in spite of just comparing with the CFD average from other channels. This indicates that neural networks can be successfully used to improve precision at no additional hardware cost. The only tradeoff is the increased complexity of neural networks compared to CFD. However, once trained, they can be used as a black box in the same way as CFD. Additionally, the training procedure can be automated, which reduces the level of expert knowledge required to use neural networks to a minimum.

It is important to keep in mind that while this thesis presents a variety of numerical experiments, the potential ways to test or improve neural networks are practically infinite, and many questions still remain unanswered. For example, hyperparameter tuning could also be repeated for a dataset composed of events from multiple fills. In this case, the network could benefit from waveform normalisation, which would reduce the spectrum of possible inputs and therefore decrease the required network capacity. Additionally, a larger network might be optimal for training using such a dataset.

[8]KUKDM 2023, 'Deep Neural Networks for the Calibration of Timing Detectors': `https://events.plgrid.pl/event/18/contributions/155/`

[9]FAST 2023, 'Using deep neural networks to improve the precision of fast-sampled particle timing detectors': `https://indico.cern.ch/event/1214183/contributions/5393328/`

Another notable finding of this research is that it is optimal to train the network on the full dataset and then use it for offline analysis. The network models used in this research achieve the best performance when trained on the data from the same run as the test one, with slightly better results of networks trained only on one channel compared to those trained on all the channels. However, the networks can also be used in an online fashion and still provide improvement, as shown in Chapter 6.6. The key would be to use a network trained only on data from recent runs, preferably from the same fill. Of course, this would not be possible for the first run in a fill, let alone the first fill in a year or after a significant change in LHC conditions. Furthermore, it might be beneficial to introduce waveform normalisation for online processing. Even if it reduces the best possible precision of a network (as shown in the results from the hyperparameter tuning procedure), the networks would certainly improve their stability and presumably work better for runs that were not used for training.

It is worth noting that although the numerical experiments included in the thesis yielded satisfactory results, not all tested ideas were successful. The most promising idea was to improve the way neural networks are trained. In a perfect scenario, training a network on the average of two channels would allow the network to provide precision based on two channels when given a waveform from only one channel as input. If this were the case, the error between the neural network and the two-channel reference would be very close to zero. Unfortunately, this was not the case, and the error was measurable. However, another network could be potentially trained using the average of timestamps acquired using the first network as new ground truth values. In the most optimistic scenario, after many iterations of this procedure, the network could be able to achieve minimal pairwise precision, as it would already predict the average of all available channels with only one channel as input. The experiments showed that this was indeed achieved and the pairwise precision was reduced with each iteration. The only issue was that the timestamps became highly inconsistent. For some waveforms, the time of arrival was predicted before the rising edge, while for others it was predicted at the end of the rising edge. This inconsistency is not desirable and unfortunately disqualifies this iterative procedure from being used in real reconstruction.

To conclude the discussion, the analysis code was written in Python using Jupyter Notebooks. All relevant notebooks and source code can be found in the project repository on GitHub[10].

## 7.3. Future work

This thesis provides a knowledge framework that could be applied in various ways for future research. Firstly, it could be utilised in the actual reconstruction of protons for CMS-PPS. This would be challenging, as it would require implementing the networks in C++, which is the main language of the software framework used for CMS event reconstruction – CMSSW. Alternatively, there may be a way to use the time of arrival timestamps computed in Python within the CMSSW offline reconstruction. However, the method of using the networks in the LHC event reconstruction has not yet been determined.

Furthermore, many questions still remain unanswered. The hyperparameter tuning procedure could be executed on multiple runs at once. It may also be possible to improve the iterative training procedure described in Chapter 7.2, resulting in more reasonable results. If these and other potential

[10]Project repository: `https://github.com/MatiXOfficial/pps-timing-ml-sampic`

research targets are investigated in the future, this thesis can serve as a solid baseline, providing valuable know-how together with already developed and implemented testing solutions. Additionally, while the hyperparameter tuning procedure was originally developed for the test beam experiment mentioned in Chapter 7.2, it was improved during the work on this thesis and can be easily used in the future.

Finally, it may also be profitable to test networks trained on different datasets with reference timestamps of higher quality, such as those used for the test beam experiment. With proper data preprocessing, such networks could potentially be used with LHC data. If successful, they could significantly improve precision, since they would not be limited by the quality of the LHC waveforms and CFD used to build the reference dataset.

# References


[1] *CERN official website*. URL: https://home.cern/ (visited on 17/02/2023).

[2] Georges Aad et al. 'Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC'. *Physics Letters B* 716.1 (2012), pp. 1–29.

[3] Serguei Chatrchyan et al. 'Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC'. *Physics Letters B* 716.1 (2012), pp. 30–61.

[4] VM Abazov et al. 'Odderon Exchange from Elastic Scattering Differences between $pp$ and $p\bar{p}$ Data at 1.96 TeV and from $pp$ Forward Scattering Measurements'. *Physical review letters* 127.6 (2021), p. 062003.

[5] Lyndon Evans and Philip Bryant. 'LHC machine'. *Journal of instrumentation* 3.08 (2008), S08001.

[6] *CERN: The third run of the Large Hadron Collider has successfully started*. URL: https://home.cern/news/news/cern/third-run-large-hadron-collider-has-successfully-started (visited on 18/02/2023).

[7] Roman Adolphi et al. 'The CMS experiment at the CERN LHC'. *Jinst* 803 (2008), S08004.

[8] Georges Aad et al. 'The ATLAS experiment at the CERN large hadron collider'. *Jinst* 3 (2008), S08003.

[9] M Albrow et al. *CMS-TOTEM precision proton spectrometer*. Tech. rep. CERN-LHCC-2014-021, TOTEM-TDR-003, CMS-TDR-013. CERN, 2014.

[10] CMS collaboration et al. 'The CMS Precision Proton Spectrometer at the HL-LHC–Expression of Interest'. *arXiv preprint arXiv:2103.02752* (2021).

[11] Giovanni Anelli et al. 'The TOTEM experiment at the CERN large hadron collider'. *Journal of Instrumentation* 3.08 (2008), S08007.

[12] Marco Oriunno et al. 'The roman pot for the LHC'. *Prepared for European Particle Accelerator Conference (EPAC 06), Edinburgh, Scotland*. 2006, pp. 26–30.

[13] Edoardo Bossini. 'The CMS Precision Proton Spectrometer timing system: performance in Run 2, future upgrades and sensor radiation hardness studies'. *Journal of Instrumentation* 15.05 (2020), p. C05054.

[14] Edoardo Bossini et al. 'Test beam results of irradiated single-crystal CVD diamond detectors at DESY-II'. CMS-NOTE-2020-007, CERN-CMS-NOTE-2020-007 (2020).

[15] M Berretti et al. 'Timing performance of a double layer diamond detector'. *Journal of Instrumentation* 12.03 (2017), P03026.

[16] Edoardo Bossini and Nicola Minafra. 'Diamond detectors for timing measurements in high energy physics'. *Frontiers in Physics* 8 (2020), p. 248.

[17] F Anghinolfi et al. 'NINO: an ultra-fast and low-power front-end amplifier/discriminator ASIC designed for the multigap resistive plate chamber'. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 533.1-2 (2004), pp. 183–187.

[18] Jorgen Christiansen. *HPTDC high performance time to digital converter*. Tech. rep. Version 2.2 for HPTDC version 1.3. CERN, 2004.

[19] E Delagnes et al. 'The SAMPIC waveform and time to digital converter'. *2014 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. IEEE. 2014, pp. 1–9.

[20] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[21] Kurt Hornik. 'Approximation capabilities of multilayer feedforward networks'. *Neural networks* 4.2 (1991), pp. 251–257.

[22] Xavier Glorot, Antoine Bordes and Yoshua Bengio. 'Deep sparse rectifier neural networks'. *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[23] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. 'Learning representations by back-propagating errors'. *nature* 323.6088 (1986), pp. 533–536.

[24] Diederik P Kingma and Jimmy Ba. 'Adam: A method for stochastic optimization'. *arXiv preprint arXiv:1412.6980* (2014).

[25] Fengxiang He, Tongliang Liu and Dacheng Tao. 'Control batch size and learning rate to generalize well: Theoretical and empirical evidence'. *Advances in Neural Information Processing Systems* 32 (2019).

[26] Nitish Shirish Keskar et al. 'On large-batch training for deep learning: Generalization gap and sharp minima'. *arXiv preprint arXiv:1609.04836* (2016).

[27] Karen Simonyan and Andrew Zisserman. 'Very deep convolutional networks for large-scale image recognition'. *arXiv preprint arXiv:1409.1556* (2014).

[28] Kaiming He et al. 'Deep residual learning for image recognition'. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[29] Olaf Ronneberger, Philipp Fischer and Thomas Brox. 'U-net: Convolutional networks for biomedical image segmentation'. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[30] Vivien Sainte Fare Garnot and Loic Landrieu. 'Panoptic segmentation of satellite image time series with convolutional temporal attention networks'. *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 4872–4881.

[31] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. *Neural computation* 9.8 (1997), pp. 1735–1780.

[32] Kyunghyun Cho et al. 'On the properties of neural machine translation: Encoder-decoder approaches'. *arXiv preprint arXiv:1409.1259* (2014).

[33] Nitish Srivastava et al. 'Dropout: a simple way to prevent neural networks from overfitting'. *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[34] Jonathan Tompson et al. 'Efficient object localization using convolutional networks'. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 648–656.

[35] Sergey Ioffe and Christian Szegedy. 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'. *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[36] Shibani Santurkar et al. 'How does batch normalization help optimization?' *Advances in neural information processing systems* 31 (2018).

[37] Edoardo Bossini. *Development of a Time Of Flight diamond detector and readout system for the TOTEM experiment at CERN*. PhD thesis. INFN, Siena, 2016.

[38] Fuyue Wang et al. 'A neural network based algorithm for MRPC time reconstruction'. *Journal of Instrumentation* 14.07 (2019), p. C07006.

[39] Fuyue Wang et al. 'The study of a new time reconstruction method for MRPC read out by waveform digitizer'. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 954 (2020), p. 161224.

[40] Fuyue Wang, Dong Han and Yi Wang. 'Improving the time resolution of the MRPC detector using deep-learning algorithms'. *Journal of Instrumentation* 15.09 (2020), p. C09033.

[41] XY Xie et al. 'A data-based machine learning approach for RPC time resolution study based on ToF reconstruction'. *Journal of Instrumentation* 16.12 (2021), P12002.

[42] Eric Berg and Simon R Cherry. 'Using convolutional neural networks to estimate time-of-flight from PET detector waveforms'. *Physics in Medicine & Biology* 63.2 (2018), 02LT01.

[43] Qi Wu et al. 'PMT Waveform Timing Analysis Using Machine Learning Method'. *IEEE Transactions on Nuclear Science* (2023).

[44] Philipp Sodmann et al. 'A convolutional neural network for ECG annotation as the basis for classification of cardiac rhythms'. *Physiological measurement* 39.10 (2018), p. 104005.

[45] Kaiming He et al. 'Delving deep into rectifiers: Surpassing human-level performance on imagenet classification'. *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[46] Muhammad Uzair Zahid et al. 'Robust R-Peak detection in low-quality holter ECGs using 1D convolutional neural network'. *IEEE Transactions on Biomedical Engineering* 69.1 (2021), pp. 119–128.

[47] Rui Cao and Guohua Liu. 'RPAA: an algorithm for R peak annotation in high-noise ECG signals based on deep learning'. *Proceedings of the 1st ACM Workshop on Mobile and Wireless Sensing for Smart Healthcare*. 2022, pp. 7–12.

[48] Jonathan J Tompson et al. 'Joint training of a convolutional network and a graphical model for human pose estimation'. *Advances in neural information processing systems* 27 (2014).

[49] Edoardo Bossini et al. 'The CMS Precision Proton Spectrometer: Precision timing with scCVD diamond crystals'. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1047 (2023), p. 167823.

[50] TensorFlow Developers. *TensorFlow*. Version v2.10.0. September 2022. DOI: 10.5281/zenodo.7604243.

[51] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[52] Hyunghun Cho et al. 'Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks'. *IEEE access* 8 (2020), pp. 52588–52608.

[53] Moein Hasani and Hassan Khotanlou. 'An empirical study on position of the batch normalization layer in convolutional neural networks'. *2019 5th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*. IEEE. 2019, pp. 1–4.

[54] Xiang Li et al. 'Understanding the disharmony between dropout and batch normalization by variance shift'. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 2682–2690.

[55] Alejandro Newell, Kaiyu Yang and Jia Deng. 'Stacked hourglass networks for human pose estimation'. *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*. Springer. 2016, pp. 483–499.

[56] Karl Pearson. 'LIII. On lines and planes of closest fit to systems of points in space'. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2.11 (1901), pp. 559–572.

[57] Laurens Van der Maaten and Geoffrey Hinton. 'Visualizing data using t-SNE.' *Journal of machine learning research* 9.11 (2008).

[58] Krzysztof Misan. *Offline reconstruction software for diamond sensors based on SAMPIC readout in the CMS-PPS detector at CERN laboratory*. AGH University of Science and Technology (PL), 2022.