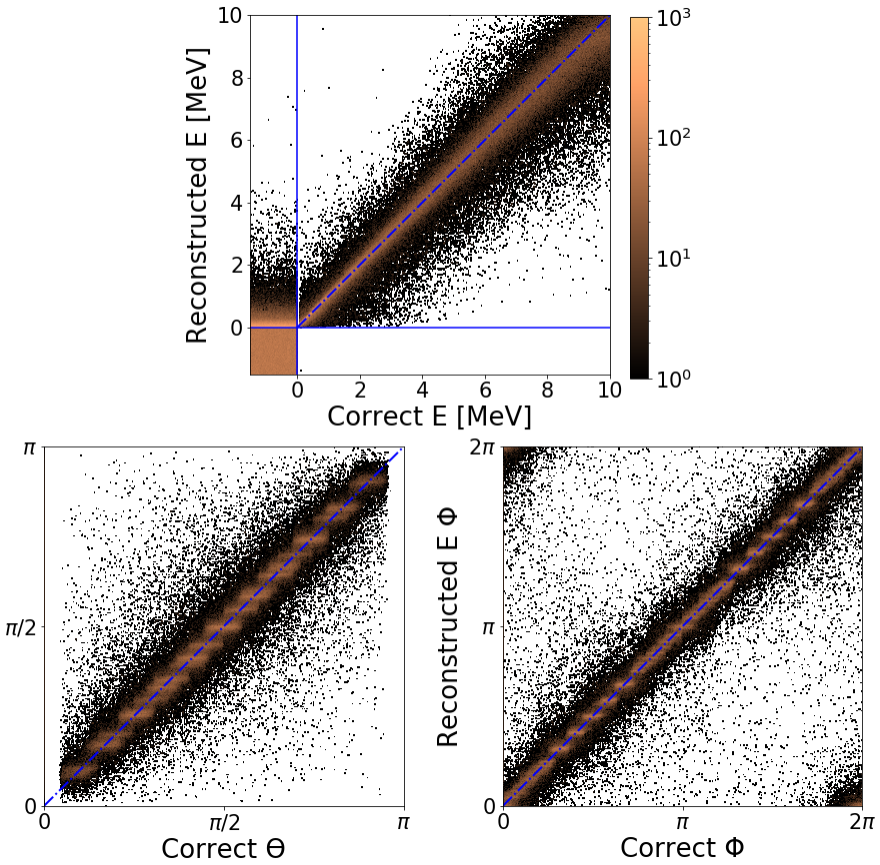
CHALMERS
UNIVERSITY OF TECHNOLOGY
|
UNIVERSITY OF GOTHENBURG



Detector reconstruction of γ-rays

An application of artificial neural networks in experimental subatomic physics

Bachelor Thesis in Engineering Physics: TIFX04-20-03

PETER HALLDESTAM
CODY HESSE
DAVID RINMAN

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Bachelor's thesis: TIFX04-20-03

Detector reconstruction of γ-rays

An application of artifical neural networks
in experimental subatomic physics

PETER HALLDESTAM
CODY HESSE
DAVID RINMAN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Division of Subatomic, High Energy
and Plasma Physics
Experimental Subatomic Physics
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Detector reconstruction of γ-rays
An application of artificial neural networks in experimental subatomic physics
Peter Halldestam
Cody Hesse
David Rinman

Supervisors: Andreas Heinz and Håkan T. Johansson, Department of Physics
Examiner: Lena Falk, Department of Physics

Bachelor's Thesis TIFX04-20-03
Department of Physics
Division of Subatomic, High Energy and Plasma physics
Experimental subatomic physics
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Reconstruction of the energy, polar angle and azimuthal angle for γ-rays with beam frame energy up to 10 MeV. For more details, see section 4.4.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Abstract

The study of nuclear reactions through measuring emitted γ-rays becomes convoluted due to complex interactions with the detector crystals, leading to cross-talk between neighbouring elements. To distinguish γ-rays of higher multiplicity, the detector data needs to be reconstructed. As a continuation on the works of earlier students, this thesis investigates the application of neural networks as a reconstruction method and compares it to the conventional addback algorithm. Three types of neural networks are proposed; one Fully Connected Network, a Convolutional Neural Network (CNN) and a Graph Neural Network (GNN). Each model is optimized in terms of structure and hyperparameters, and trained on simulated data containing isotropic γ-rays, before finally being evaluated on realistic detector data.

Compared to previous projects, all presented networks showed a more consistent reconstruction across the studied energy range, which is credited to the newly introduced momentum-based loss function. Among the three networks, the fully connected performed the best in terms of smallest average absolute difference between the correct and reconstructed energies, while having the fewest number of trainable parameters. By the same metric, none of the presented networks performed better than addback. They did, however, show a higher signal-to-background ratio in the energy range of 3–6 MeV. Suggestions for further studies are also given.

Keywords: artificial neural networks, convolutional neural networks, graph neural networks, gamma ray reconstruction, addback, Crystal Ball, TensorFlow, Keras

Sammanfattning

Då joner accelereras och kollideras med ett strålmål, kan de studeras bl.a. genom att mäta den emitterade γ-strålningen. Sådana experiment försvåras dock av γ-strålningens växelverkan i detektorerna, något som leder till överhörning mellan närliggande detektorenheter. För att särskilja strålning av högre multiplicitet krävs därför en rekonstruktionsmetod, vilken konventionellt varit addback-algoritmen. Denna rapport är en uppföljning på föregående års kandidatarbeten, och undersöker tillämpningen av artificiella neurala nätverk som ett alternativ till addback. Tre typer av neurala nätverk presenteras; ett fullt anslutet nätverk, ett faltande nätverk, samt ett grafnätverk. Vardera modell optimeras både i termer av hyperparametrar och nätverksstruktur, och tränas på simulerad data i form av isotropisk γ-strålning. Slutligen utvärderas nätverken på mer realistisk data.

Samtliga nätverk uppvisar en mer konsekvent rekonstruktion över det studerade energiomfånget jämfört med föregående års nätverk, något som tillskrivs den nya rörelsemängdsbaserade kostnadsfunktionen. Bland de prövade nätverken presterade det fullt anslutna bäst i minsta genomsnittliga absolut skillnad mellan rekonstruerad och korrekt energi, samtidigt som det innehöll minst antal träningsbara parametrar. Addback kvarstod dock som bäst i dessa mått, medan nätverken uppvisade ett bättre signal-bakgrundsförhållande i intervallet 3–6 MeV. Vidare presenteras förslag för fortsatta studier.

Acknowledgements

First of all we would like to state our gratitude towards our supervisors Andreas Heinz and Håkan T. Johansson, both for presenting us with this project, and for offering their guidance as to see it through. The task has been equal parts challenging, as intriguing and educational, and could not have been done without your advice.

We would also like to thank our predecessors J. Jönsson, R. Karlsson, M. Lidén, R. Martin as well as J. Olander, M. Skarin, P. Svensson, and J. Wadman for laying a foundation for us to work upon. Building upon what you have accomplished has been a luxury, and we would not have come as far without you.

A special thanks also goes out to the bright minds at Google for developing the machine learning frameworks TensorFlow and Keras, allowing us to focus on building our models rather than for the code to compile.

Lastly, we would like to thank the Swedish National Infrastructure for Computing (SNIC) and High Performance Computing Center North (HPC2N) for granting us the resources to train our networks without ever having to worry about running out of either memory or time.

The authors, Gothenburg, May 2020

Contents

Chapter 1

Introduction

The fascination of better understanding one's place in the universe is driving us to seek answers to one of the most fundamental questions: What are we *made of*? In nuclear physics, one way to approach this question is by the study of nuclear reactions with particle accelerators. This thesis aims to investigate the possibility of using *neural networks* in the reconstruction of energies and emission angles of γ-rays in such experiments.

1.1 BACKGROUND

When a beam of ions is accelerated to relativistic velocities and collides with a stationary target, there is a possibility of energy being released in the form of γ-radiation. This energy can be measured using an array of detectors surrounding the target, thus revealing some properties of the studied nuclei.

One detector designed for this purpose is the Darmstadt-Heidelberg Crystal Ball. Consisting of 162 scintillating NaI-crystals arranged in a sphere as illustrated in Fig. 1.1, it is capable of measuring both the energies and angles of incident γ-rays.

However, due to cross-talk between neighbouring detector elements, mainly due to Compton scattering, it can in some cases be difficult to correctly assign higher multiplicities of photons [1]. Because of this, the data from the detector needs to be carefully reconstructed in order to determine whether the energy of a single photon was deposited in several crystals or not.

A method of dealing with this reconstruction problem is a set of algorithms, known as the *addback*-routines. They have previously been investigated using simulations [2], and were found to correctly identify around 70% of 1 MeV γ-rays. This accuracy, however, rapidly decreases for higher energies.

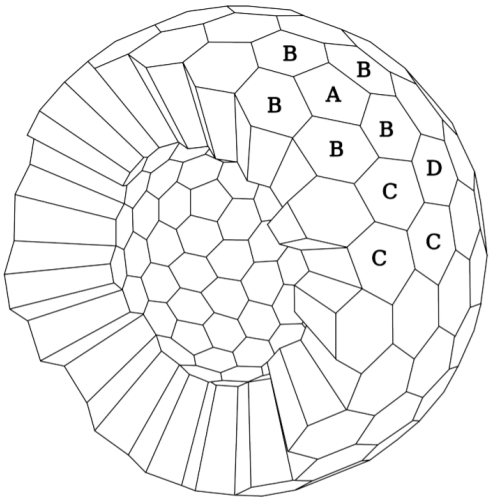


Figure 1.1: *Cross-sectional diagram of the Crystal ball with its 4 shapes of crystal detectors.* [1]

Due to the complexity of the γ-reconstruction, the question whether a neural network could outperform addback at this task is raised. Neural networks have seen great development in recent years and are being implemented more and more in almost every field of science and technology. Ranging from deep neural networks, praised for their good approximation of complicated relationships, to convolutional neural networks, which excel at computationally efficient pattern recognition; they all seem fit to offer a competetive alternative to the addback algorithms.

1.2 PREVIOUS WORK

This thesis is based upon two previous bachelor projects at the Department of Physics, Chalmers University of Technology, Gothenburg. It is a continuation aiming to evaluate the application of machine learning in γ-ray event reconstruction. In 2018, Olander *et al.* [3] investigated this question and obtained results comparable to the addback algorithm. However, their results show artefact in the reconstruction of photon energies and angles, as well as in their signal-to-background ratio comparsion to addback. In 2019, Karlsson *et al.* [4] further developed the same concepts, focusing on a different structure of machine learning: convolutional neural networks and reaching similar results to those of Olander *et al.* .

1.3 PURPOSE AND AIMS

The goal of this project is to further investigate the application of machine learning in the reconstruction of energies and emission angles of γ-rays for the Crystal Ball detector. In particular, the is aim to study and optimize different types of neural networks including *Fully Connected Networks* (FCN), *Convolutional Neural Networks* (CNN) and *Graph Neural Networks* (GNN) in order to assess their performance at this task, and whether they can achieve a more accurate reconstruction than the addback algorithm.

1.4 DELIMITATIONS

Even though newer and more advanced particle detectors such as CALIFA [5] exist today, we will only be performing tests on the Crystal Ball detector. This is due to its simpler geometry, in order to provide a proof of concept. The results should, however, be generalisable to other similar detectors, including CALIFA [3].

Since the nuclei in the accelerated beams travel at 50–70% of the speed of light, it is necessary to apply relativistic corrections to the detector data in order to study the nuclear collisions in their proper frame of reference. Both Olander *et al.* [3] and Karlsson *et al.* [4] have dealt with such corrections directly in the loss function (see section 3.1.2), which was found to improve the performance of the neural networks. However, this limits the applicability of the networks, as they have to be retrained in order to accommodate for different beam velocities. Furthermore, the relativistic corrections consist of rather simple calculations that can easily be applied to the reconstructed data. Therefore, this study will be limited to reconstructions in the laboratory frame.

Lastly, this thesis focuses mainly on the case where there is, at most, two coinciding γ-rays in the initial reaction. This is to shorten the time required for each neural network to be trained. The software is however generalizable, and thus capable of dealing with higher orders of maximum multiplicities, as briefly discussed in sec. 4.5.1.

Chapter 2

Detection of γ-rays

The underlying problems investigated in this thesis encompass a multitude of subjects, including machine learning, data simulation and subatomic physics. This chapter focuses on the latter, introducing the necessary underlying concepts of detector physics and the most commonly used set of analysis methods.

2.1 THE CRYSTAL BALL

The Darmstadt-Heidelberg Crystall Ball is a detector that was built in collaboration between GSI Darmstadt, the Max-Planck-Institute for Nuclear Physics and the University of Heidelberg. The detector is made of 162 scintillating NaI-crystals forming a sphere with an inner radius of 25 cm enclosing the target, only leaving openings for the radioactive beam to pass through.

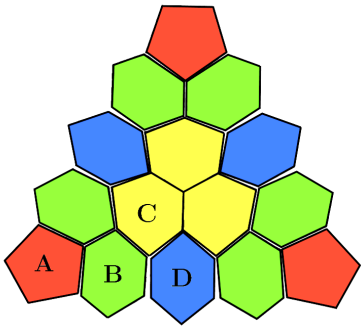


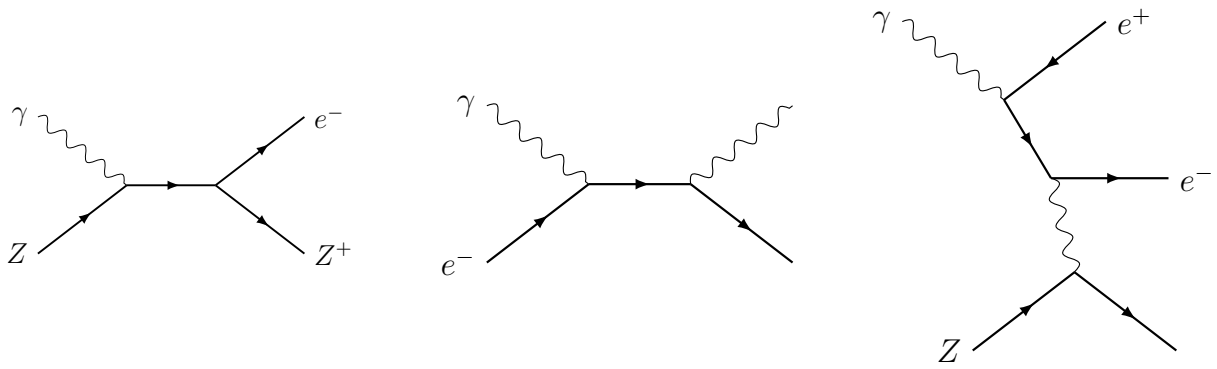
Figure 2.1: *Arrangement of the four shapes of detector elements in the Crystal ball. This is the projection of one of 20 equilateral spherical triangles, that together constitute the 162 detectors* [1].

The crystals cover an equal solid angle of 0.076 sr each, and in total 98 % of the full solid angle [1]. They come in 4 different shapes; regular pentagons called A crystals and three different kinds of irregular hexagons called B, C, and D crystals. The arrangement of these crystals, resulting in a spatial resolution of ± 8 degrees, can be seen in Figs. 1.1 and 2.1.

2.2 INTERACTIONS OF γ-RAYS WITH SCINTILLATORS

A beam of γ-radiation passing through matter decreases exponentially in intensity while any individual γ-ray loses energy in discrete steps/interactions, unlike the gradual behaviour of charged particles such as protons and electrons. This mostly occurs in three possible ways, at least in the energy range of interest [6]: the *photoelectric effect*, *Compton scattering* and *pair production*.

The photoelectric effect is the emission of a charged particle, often e^-, due to the total energy absorption of γ-rays as schematically shown in Fig. 2.2a. The kinetic energy of the emitted particle is the difference in the energy of the photon and



(a) Photoelectric effect **(b)** Compton scattering **(c)** Pair production

Figure 2.2: *Feynman diagrams showcasing the three main processes by which photons interact with matter. The first (a) shows an electron being released after absorbing a photon, and Compton scattering is shown in (b). The rightmost diagram (c) illustrates the production of an $e^{\pm}$-pair near a nucleus, marked as Z.*

the binding energy of the electron.

The scattering of γ-rays by charged particles, again usually an e^-, is what is commonly referred to as Compton scattering. Only a part of the photon energy is transferred to a recoiling electron leaving a less energetic γ with a higher wavelength.

Finally, pair production, which is the creation of a subatomic particle and its anti-particle from the annihilation of a neutral boson. In this case it is

$$\gamma \longrightarrow e^+ + e^-,$$

i.e. an $e^{\pm}$-pair created from a γ as illustrated in Fig. 2.2c. Due to conservation of momentum, this can only occur within the close proximity of a nucleus.

Of these three processes, at the energy range of interest (100 keV–20 MeV), Compton scattering is usually the dominating way of interaction with the NaI-*scintillators* [4].

Scintillators such as those in the Crystal Ball absorb the energy from an incoming particle and then re-emits it in the form of visible light. The emitted light intensity from the scintillators is approximately proportional to the deposited energy. To detect any ionizing γ-radiation[1], these signals are detected and amplified with a photomultiplier.

2.3 RELATIVISTIC EFFECTS

The beams used to study colliding nuclei at the Crystal Ball usually travel at high speeds, $\beta = 0.5 - 0.7$ in terms of the speed of light; however the γ-rays are measured in the laboratory frame of reference. This means that relativistic effects must be taken into account. These are the relativistic Doppler effect and headlight shift, where the latter refers to the fact that the emitted γ-rays appear to be scattered conically in the forward direction [3]. The γ-ray energy E and polar emission angle θ in the laboratory frame relates to E', θ' in the beam's center of mass frame of reference [7, p. 78–

[1] Works with α and β radiation as well.

82] as

$$E' = \frac{1-\beta\cos\theta}{\sqrt{1-\beta^2}}E, \qquad \cos\theta = \frac{1+\cos\theta'}{1+\beta\cos\theta'}. \tag{2.1}$$

2.4 DATA ANALYSIS USING ADDBACK ROUTINES

When an energetic γ-ray hits a detector in the crystal ball, part of its energy might be deposited in the surrounding crystals due to scattering and other nuclear interactions mentioned in section 2.2. The scattered γ-rays can, in turn, re-scatter resulting in a chain of detector interactions, hence making the reconstruction of a single γ-ray a demanding task. The classical solution is to implement an addback algorithm, typically carried out by taking the sum over the energies measured by a group of adjacent detectors, often referred to as a *cluster* [2, 8].

In order to identify a cluster, and to find the energy deposited from each particle interaction, it is usually assumed that the first interaction will deposit the largest proportion of the total deposited energy, $E_{\max}$ [8]. Scattered γ-rays are likely to interact with neighbouring crystals and may re-scatter, depositing energies E'_i and E''_j respectively.

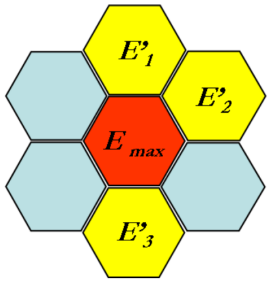
The most commonly-used addback routines are *Neighbour* and *Second Neighbour*. As the name implies, the Neighbour algorithm calculates the sum of the initial hit and its direct neighbours, ignoring energies deposited from re-scattering, as

$$E_{\text{tot}} = E_{\max} + \sum_i E'_i.$$

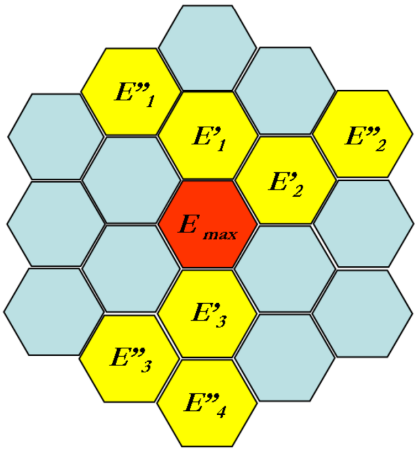
where i are the neighbours to the central crystal. Second Neighbor is expanded in order to include second-ring crystals j to account for chain-scattering events. The total energy is then calculated as,

$$E_{\text{tot}} = E_{\max} + \sum_i E'_i + \sum_j E''_j.$$

These two addback routines are illustrated in fig. 2.3.



(a) *First neighbour*



(b) *Second neighbour*

Figure 2.3: *Schematic representations of the two clusters containing the crystal neighbours (a) and the second-neighbours (b) with respect to the initial hit* [8].

Chapter 3

Artificial neural networks

Artificial neural networks (ANN) have in recent years become an integral part of modern society, making its way into our everyday lives in trivial matters such as individual targeted recommendations of music and movies. As its computational prowess is used to model a multitude of essential processes: ranging from the natural sciences to economics and politics. The following chapter aims to give a brief introduction to the extensive subject of ANNs.

Table 3.1: *Notation used when describing the key mathematical aspects of artificial neural networks. These are introduced in this chapter and used throughout this thesis.*

Notation	Description
$\boldsymbol{x}$	Input
$\boldsymbol{y}$	Network output
$\hat{\boldsymbol{y}}$	Correct output
$\boldsymbol{h}^{(\ell)}$	Hidden layer of index ℓ
$W^{(\ell)}$	Weight matrix
$\boldsymbol{b}^{(\ell)}$	Bias vector
D	Depth (# hidden layers)
N_ℓ	Width (# neurons per layer)
$\mathcal{A}$	Activation function
$\mathcal{L}$	Loss function

3.1 BASIC CONCEPTS

An ANN are a computing system capable of modelling various types of different non-linear problems. Mathematically, they can be thought of as an approximation f of some function $\hat{f} : \boldsymbol{x} \mapsto \hat{\boldsymbol{y}}$, where $\boldsymbol{x}$ is some input feature data and $\hat{\boldsymbol{y}}$ the correct output. This approximation is iteratively refined in a processes called *training*, to be discussed in sections 3.1.2 and 3.1.3.

Getting its name from the analogy to the human brain, an ANN is comprised of fundamental components commonly referred to as *neurons*, each containing a number called its *activation*. These neurons are interconnected to form a network, which is primarily defined by its structure, i.e. how the neurons are connected, but also from *hyperparameters* defining certain quantities and operation conditions of the networks, allowing for tuning of the model.

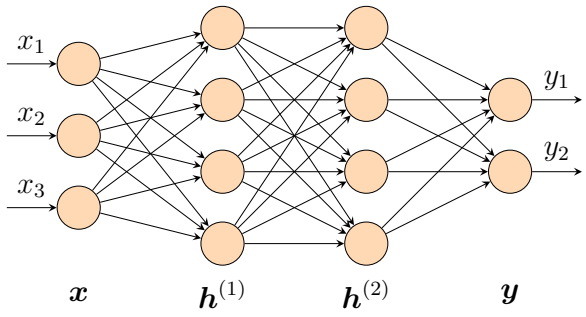


Figure 3.1: *Diagram of a simple fully connected network with two hidden layers (depth two), showing the relationship between the neurons. The interconnecting arrows represents the elements in the three weight matrices $W^{(1)}$, $W^{(2)}$ and $W^{(3)}$.*

3.1.1 Forward propagation

A common network structure design is to group the neurons into subsequent layers with every neuron connected to each neuron in the layers before and after. This is known as a *Fully Connected Network* (FCN) and an example of a small[1] such network is illustrated in fig. 4.19. These layers can be represented by vectors containing the activations for each of its neurons. The first layer of the network serves as the input $\boldsymbol{x}$, which is connected to the output $\boldsymbol{y}$ through a network of hidden layers $\boldsymbol{h}^{(\ell)}$. The superscript ℓ denotes the layer index such that $\boldsymbol{h}^{(0)} = \boldsymbol{x}$ and $\boldsymbol{h}^{(D+1)} = \boldsymbol{y}$. The *depth* D is the total number of hidden layers, and similarly the amount of neurons in each layers is called its *width* and is denoted with N_ℓ. Both of these quantities are referred to as hyperparameters. Consider a hidden layer $\boldsymbol{h}^{(\ell)}$. Its state is determined by the layer before it from the forward propagation rule

$$\boldsymbol{h}^{(\ell)} = \mathcal{A}\Big(W^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\Big), \tag{3.1}$$

where W is the *weight matrix* and $\boldsymbol{b}$ the *bias vector*, which both are trainable parameters, discussed in sec. 3.1.3. This is commonly referred to as a *Dense* connection. The function $\mathcal{A}$ acting elementwise is called the *activation function* and is the key component in any ANN. If these are elements of the linear functions, the network f would completely reduce to some linear functions, capable only of solving linear tasks.

To successfully apply networks to non-linear tasks, non-linear activations are needed. Perhaps the most common activation is the *Rectified Linear Unit*, given by

$$\text{ReLU} : \; x \longmapsto \max(0,\, x). \tag{3.2}$$

This is default activation used throughout this thesis, unless otherwise mentioned.

Another commonly used activation function is the *Sigmoid function* given by

$$\sigma : \; x \longmapsto \frac{1}{1 + e^{-x}},$$

which has the characteristic of producing values in the interval between 0 and 1. This makes it suitable for output neurons expected to represent some probability as explored in section 4.8.

3.1.2 The loss function

In order for a neural network to function, it must first be trained. The process of supervised training in the context of FCN:s refers to optimizing the set of weights and biases $\{W^{(\ell)}, \boldsymbol{b}^{(\ell)}\}$ which minimizes the so called *loss function*[2] over a given set of training data $\{\boldsymbol{x}, \hat{\boldsymbol{y}}\}$. The loss function, denoted $\mathcal{L}$, is a metric of how well the output of the network $\boldsymbol{y}$ corresponds to the correct labels of the training set $\hat{y}$.

There are a multitude of different loss functions, some more suitable for certain kind of tasks. In regression problems, where

[1] In relative terms compared to those examined in this thesis (see section 4.5).

[2] Other commonly used names are *cost function*, *error function* and *objective function*.

$\hat{f}$ is a continuous distribution, the standard choice of $\mathcal{L}$ is the Mean squared error defined as:

$$\mathrm{MSE}(\boldsymbol{y},\ \hat{\boldsymbol{y}}) \equiv \frac{1}{n} \left\| \boldsymbol{y} - \hat{\boldsymbol{y}} \right\|^2 \tag{3.3}$$

where $n = N_{D+1}$ is the number of neurons in the output layer.

Another type of regression problems are those of a binary classification, i. e. true or false. The question could for example be: is this a picture of a cat? The standard choice in this case would be the *Binary Cross-Entropy* H, given by

$$H(\beta,\ \hat{\beta}) \equiv -\hat{\beta} \log \beta - (1 - \hat{\beta}) \log(1 - \beta)$$

where $\hat{\beta} \in \{0, 1\}$ is the correct binary value and $0 < \beta < 1$ is the output of the network.

3.1.3 Backpropagation

According to eq. (3.1) the output layer $\boldsymbol{y} = \boldsymbol{h}^{(D+1)}$ depends on every preceding layer and the corresponding weights and biases of the network. Optimization is thus done by first calculating the gradient of $\mathcal{L}$ with respect to these parameters and then updating them through gradient descent by

$$\theta \longmapsto \theta - \eta \nabla_\theta \mathcal{L},$$

where θ is any such trainable parameter and η is a hyperparameter called the *learning rate*. This procedure is not restricted only to FCN:s and the general principle is the same also for other types of propagation rules than those in eq. (3.1).

While these gradients may be calculated analytically, they can be computationally expensive to evaluate due to the non-linear activation functions. Because of this, it is often preferred to use the *backpropagation* algorithm, which instead relies on the chain rule to produce good approximations of $\nabla_\theta \mathcal{L}$.

The computational complexity can be reduced further by dividing the training set into smaller batches and estimating the gradient using the average value of the loss function over each batch. This method of minimizing the loss function subject to the weights and biases of the network, i.e. training the network, is referred to as *Stochastic Gradient Descent* [9] and is the principle upon which the optimizer used for this thesis, called *Adaptive Moment Estimation*[3] [10], is based.

During training, the entire training data set is presented to the network several times, where each pass is called an *epoch*. The training data only consists of a portion of the total data, with the remainder used as a validation set. This enables a good metric for calculating the performance of the network called *validation loss*, which is the loss function calculated over the validation set. Should the validation loss increase over several epochs, whereas the training loss decreases, the network is being over-trained, i.e. overfitted to the training data. To prevent this, one can implement *early stopping*, which automatically halts the training whenever the validation loss stops decreasing over a set number of epochs, called *patience*.

3.2 CONVOLUTIONAL NETWORKS

A *Convolutional neural network* (CNN) is a neural network implementing one or several *convolutional layers*, usually followed by a FCN. Convolutional neural networks excel at local feature recognition, making them widespread in machine learning applications such as image recognition.

A convolutional layer applies a mathematical operation on an input tensor in order to produce a number of output tensors.

[3] Often abbreviated as ADAM

Focusing on the one-dimensional case, as used in this thesis, the input of one operation consists of a vector and produces a number of vectors, or *feature maps*, specified by a hyperparameter. Associated to each feature map is a smaller vector consisting of trainable parameters called a *kernel* (or filter). The elements in each feature map is produced by *convolving* the kernel with the input, i.e. calculating the sum of each element in the kernel multiplied with the respective value of the input, as seen in Fig. 3.2. Here, a uniform bias can also be added to the output. To produce the next element of the feature map, the kernel is shifted by a number of indices over the input matrix given by a hyperparameter called the *stride*.

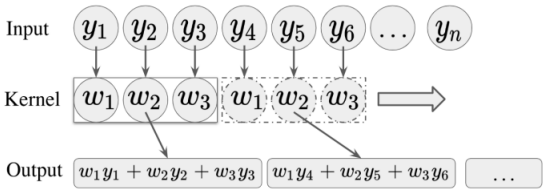


Figure 3.2: *Visualization of how the output of a 1-dimensional convolutional layer is formed. This layer features a stride and filter size of 3. The application of bias is not shown* [4].

The stride should be smaller than or equal to the dimension of the kernel in order for the convolutions to include each value of the input at least once. However, there are several design choices to consider. When choosing a larger stride relative to the kernel size, one effectively downsamples the feature map, thus reducing the number of trainable parameters. In contrast, choosing a smaller stride can result in better recognition of details, i.e. larger sensitivity to variations over fewer indices of the input. Another thing to account for is that a stride strictly smaller than the kernel size always results in values along the edges of the input matrix contributing less to the resulting feature map than those in the center. This can be avoided by utilizing *padding*, which is the concept of introducing an interval of zeroes at the start and end of the input vector. If the length of this interval is such that the resulting feature map has the same dimension as the input, this is called *Same Padding*. On the contrary, *Valid Padding* means that no padding is applied.

As mentioned above, a larger stride downsamples the data and therefore results in less trainable parameters. In a CNN, downsampling can also be achieved by applying *pooling*. A pooling layer collects the information of the previous layer by, similar to the convolutional layer, applying a kernel over the input and calculating the corresponding value of the output according to some rule. Common rules to use for pooling is either *max pooling* or *average pooling*, meaning the kernel selects the maximum or average value from its overlap with the input vector, respectively. The stride of the pooling kernel is always equal to its size, hence the pooling operation produces an output downsampled by a factor equal to the kernel size [9].

3.3 GRAPH NETWORKS

In the last decade a whole new subgenre of ANNs has been developed, called *Graph neural networks* (GNN). First introduced by Scarselli *et al.* [11], these networks process data in graph domains. A graph $\mathcal{G}$ is a data structure comprising a set of nodes and their relationships to one another. Currently, most GNNs have a rather universal design in common, and are distinguished only by how a layer $\boldsymbol{h}^{(\ell)}$ is aggregated from this graph.

An adjacency matrix A is a representation of any finite $\mathcal{G}$. Its binary elements indicate whether pairs of nodes are adjacent or not. For the purpose of this thesis, $\mathcal{G}$ is an undirected graph and therefore, A is diagonally symmetrical.

Each network layer $\boldsymbol{h}^{(\ell)}$ is aggregated as $\boldsymbol{h}^{(\ell)} \mapsto \boldsymbol{a}(\boldsymbol{h}^{(\ell)}, A)$. he next layer in the forward propagation can thus be described by

$$\boldsymbol{h}^{(\ell+1)} = \mathcal{A}(W^{(\ell+1)}\boldsymbol{a} + \boldsymbol{b}^{(\ell+1)}).$$

Different GNNs vary in the choice of the parametrization of $\boldsymbol{a}$. One such choice is the *spectral rule* (3.4) as described by T. N. Kipf and M. Welling [12]. The aggregation takes the form

$$\boldsymbol{a}(\boldsymbol{h}^{(\ell)}, A) = D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}\boldsymbol{h}^{(\ell)}, \tag{3.4}$$

where D is the diagonal degree matrix of $\mathcal{G}$ and $\tilde{A} = A+I$ (I being the identity matrix). The latter is needed in order to include self loops in the aggregation. This is similar to the Dense connection eq. (3.1), with the only difference being in the aggregation of $\boldsymbol{h}^{(\ell)}$. This it is referred to as a *GraphDense* connection.

Let us consider the aggregation of the i:th node in the hidden layer $\boldsymbol{h}^{(\ell)}$. The spectral rule eq. (3.4) gives

$$\begin{aligned} a_i(\boldsymbol{h}^{(\ell)}, A) &= \left(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}\boldsymbol{h}^{(\ell)}\right)_i \\ &= \sum_n D_{in}^{-\frac{1}{2}} \sum_j \tilde{A}_{ij} \sum_m D_{jm}^{-\frac{1}{2}} h_j^{(\ell)} \\ &= \sum_j \frac{\tilde{A}_{ij}}{\sqrt{D_{ii}D_{jj}}} h_j^{(\ell)}, \end{aligned}$$

where the final simplification comes from D being diagonal. This can roughly be thought of as the sum of $h_i^{(\ell)}$ and its neighbouring nodes, which explains the inclusion of the identity matrix in $\tilde{A}$[4].

In conclusion, a network using these principles has the advantage of having relationships of every node, i.e. neuron, known from the start. This could potentially reduce the time of training since a non-GNN would need to learn these relationships.

3.4 BATCH NORMALIZATION

It has been shown that the training rate of a single neuron is greatly improved if every element in $\boldsymbol{h}^{(\ell)}$ is normalized giving them a mean value of zero and unit variance [14]. In ANNs each layer $\boldsymbol{h}^{(\ell)}$ gets it trainable parameters updated proportional to the partial derivative of the loss function of the following layer $\boldsymbol{h}^{(\ell+1)}$. Hence, if the outputs from two layers differ greatly in value, the gradient from the smaller output may become vanishingly small in comparison, which is often referred to as the *vanishing gradient problem*. This may effectively prevent certain neurons from changing their values and, in the worst case, completely stop the neural network from further training.

The normalization thus ensures that the activations for all neurons are of the same order of magnitude, which gives them equal opportunity to influence the learning of their connected neurons. For a layer of width N, each element $h_i^{(\ell)}$, can be normalized as

$$h_i^{(\ell)} \longmapsto \frac{h_i^{(\ell)} - \mathbb{E}[h_i^{(\ell)}]}{\sqrt{\mathrm{Var}[h_i^{(\ell)}]}} \tag{3.5}$$

where the expectation, $\mathbb{E}[h_i^{(\ell)}] \approx \mu_b$, and variance, $\mathrm{Var}[h_i^{(\ell)}] \approx s_b^2$, are estimated with the mean μ_b and sample standard deviaton s_b over a batch denoted b of training data, hence the name *batch normalization* [15].

The process of normalization presented thus far can however change, or even limit, what the layer can represent. For instance, normalizing the inputs of the sigmoid function will limit it to be approx-

[4]Without it, the aggregation of the i:th node would just be the mean of its neighbours exluding the node itself! For the interested reader: a more comprehensive explanation behind the normalization using the degree matrix can be found in [13]

imately linear. The power of representation is easily regained by ensuring that the transformation inserted in the neural network can represent the identity transform. This is achieved by introducing two trainable parameters for each layer to be normalized: one for scaling, and another for shifting, allowing it to take on any range of numbers.

Chapter 4

Method development

The core of this thesis lies in exploring various methods using neural networks and the investigation of how well they perform event reconstruction of γ-rays. This chapter presents the key aspects of the procedural development of such methods and the resulting set of networks selected to be compared to the addback routine in the following chapter. Although a fair amount of effort was put into the analysis of e.g. FCNs and CNNs, perhaps the most noteworthy aspect is the redesigned loss function described in section 4.3.2.

4.1 BUILDING THE NEURAL NETWORKS

The objective of these networks is to reconstruct the energies and directions for each of the γ-rays in the initial reaction from a set of detected energies $\{E_i\}$. In the case of the Crystal Ball detector, the input consists of values for each of the $N_0 = 162$ crystal detectors. Every network is trained on computer-generated data, to be discussed in sec. 4.2.

All neural networks have been implemented using *Keras* [16], a Python API running on top of Google's machine learning framework *Tensorflow* [17]. This allows us to work at a high level, using Keras' predefined enviroments for common features of ANNs (described in section 3), while utilizing the hardware-optimized computational backend of Tensorflow. Keras describes the structure of a neural network through the abstract `Model`-class, which is implemented to create the different network structures described in sections 4.5 and 4.6. The resulting product can be found at `https://github.com/PajterMclovin/DetNet`.

The majority of the network training was carried out using one of two local machines; one equipped with a Nvidia GTX 1060 GPU and the other with a 1080 Ti, both machines operating on quad-core 6th generation Intel Xeon processors and 32 GB of RAM. For final training and evaluation of convolutional networks, the HPC system Kebnekaise at HPC2N, Umeå University, Sweden, as as part of Swedish National Infrastructure for Computing (SNIC) [18], was utilized. These tests ran using nVidia K80 GPUs on nodes with 14-core 5th generation Xeon processors and 64 GB of RAM.

4.2 DATA GENERATION THROUGH SIMULATION

The training of an ANN requires an extensive pre-labeled data set. For the purposes of this study, such data sets must contain the original and measured energies in every detector crystal of the Crystal Ball for each

individual event. This is generated using a C++ based toolkit called *GEANT4*, a state of the art tool to model the interactions of particles with matter. Modeling these interactions, however, is a difficult task due to the large number of coupled degrees of freedom and the quantum-mechanical, i.e probabilistic, nature of the processes. Hence, *GEANT4* utilizes Monte Carlo simulations to produce numerical solutions [19].

The final modeling was performed using *ggland*, a wrapper program built around the *GEANT4* library, which allowed for the simulation of high-energy particle events within the Crystal Ball geometry [20]. The ggland simulations are constructed using an arbitrary number of particle sources, referred to as *guns*, placed at the center of the geometry. Each gun simulates the emission of a single particle, with an energy and angle which is randomized within the boundaries of a predetermined distribution, e.g. to emulate the scattering from target nuclei used in experiments.

To handle the large amounts of data produced within the ggland simulations, a C++ framework for data processing called ROOT was used. The ROOT-framework was originally developed by CERN in order to easily save, access and handle the petabytes of data generated in accelerator-based physics experiments [21].

For a given *maximum multiplicity* m, i.e. the maximum number of γ-rays emitted during an event, there are $m + 1$ subsets of the correct label set $\hat{\boldsymbol{y}}$. One that represents the case where there are no γ and $\hat{\boldsymbol{y}}$ is all zeros, and the rest containing between one and m γ-rays. Since the simulated data does not include the former, these are subsequently added in a given amount, see Fig. 4.1.

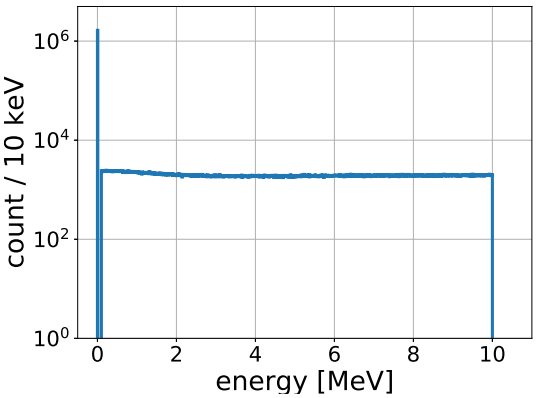


Figure 4.1: *Semi-log plot showing the energy distribution from the GEANT4 simulations which was used for training the networks. It is uniform in the interval* $0.01 - 10\,\text{MeV}$ *with an additional number of about* 10^6 *empty events.*

4.3 LOSS FUNCTIONS

The loss function is undeniably a critical part of the training of a neural network, and needs to be treated accordingly. This section explains some different approaches in defining $\mathcal{L}$ that were investigated during the development of this thesis. Ultimately the so-called photon momentum loss was selected, a rather natural definition using the momentum vectors of the γ-rays in cartesian coordinates.

4.3.1 Investigating different loss functions

During early development of the neural networks in the Keras enviroment, a number of loss functions were implemented. Primarily the non-relativistic modified MSE-loss functions of previous studies [3, 4] were tested. One such MSE variant is shown in eq. (4.1), where $\Delta E_i = E_i - \hat{E}_i$ and analogous for the angles. Note variables marked with a hat represent the correct label, whereas those without are the output of the network. The coefficients labeled λ are used to combine the different terms and were optimized by Karlsson *et al.* [4], and their final choice for the equation above

$$\mathcal{L}\{E_i,\ \theta_i,\ \phi_i\} = \frac{1}{m}\sum_{i=1}^{m}\left[\lambda_E(\Delta E_i)^2 + \lambda_\theta(\Delta\theta_i)^2 + \lambda_\phi\Big((\Delta\phi_i + \pi) \bmod 2\pi - \pi\Big)^2\right] \tag{4.1}$$

$$\mathcal{L}\{E_i,\ \theta_i,\ \phi_i\} = \frac{1}{m}\sum_{i=1}^{m}\left[\lambda_E(\Delta E_i)^2 + \lambda_\theta(\Delta\theta_i)^2 + \lambda_\phi\Big(1 - \cos\Delta\phi\Big)\right] \tag{4.2}$$

was to set all $\lambda = 1$. Whether this holds for other network structures is a different question, which was not investigated in this work. They also showed in their plots that this cost function in combination with their data sets produced artefacts, mainly in the spatial reconstruction. The polar angle θ tended to be reconstructed too low, whereas the azimuthal angle ϕ was reconstructed too low for $\phi < \pi$ and too high for $\phi > \pi$.

When implementing the same loss function in Keras, the artefacts persisted. This is expected, as the artefacts likely are a result of many different inconsistencies in the treatment of the angles and not due to the computational framework. For instance, consider an event with perfect reconstruction for the polar angle $\Delta\theta = 0$ but with $\Delta\phi = \pi$. The spatial part of the function would in this case yield a cost of $\lambda_\phi\pi^2$, regardless of θ. This is bad, because for $\theta = 0$ this would be a perfect reconstruction, wheras for $\theta = \pi/2$ it would be the worst possible. Because the coordinates are interdependent in terms of distance, a means of addressing this issue could be to weigh the azimuthal term with $\cos\theta$ as suggested by Karlsson et al., making the expression look more and more like a transformation to cartesian coordinates. This, among other things, inspires the use of a momentum loss function which will be discussed in section 4.3.2.

Another reason for the artefacts in the azimuthal angles when using the loss function in eq. (4.1) was suspected to stem from the use of the modulo function. Being a non-continuous function, it is not a preferred choice in contexts of optimization with gradient descent. Because of this, a continuous replacement as seen in eq. (4.2) was tested.

Unlike the modulo function this expession has a continuous gradient everywhere. Furthermore, the gradient of this function also points towards the closest minima, which was thought to be another advantage of this loss function. However, through testing it became clear that this function produced artefacts similar to those of eq. (4.1), likely due to the separate treatment of the angles.

4.3.2 Photon momentum loss

Since the energy and direction of a photon can be described by its momentum vector $\boldsymbol{p}$, it is a natural choice of quantity for the network to reconstruct. Continuing along this path, $\boldsymbol{p}$ is here represented using natural units ($E = \|\boldsymbol{p}\|$, $c = 1$). Let m be the maximum multiplicity. The output of the network thus takes the form of

$$Y = (\boldsymbol{p}_1,\ \ldots \boldsymbol{p}_m) \in \mathbb{R}^{3m}.$$

In practice, each vector can be identified with three cartesian coordinates, hence $Y \in \mathbb{R}^{3m}$. For the case $m = 2$, the output would be

$$Y = (p_{1x},\ p_{1y},\ p_{1z},\ p_{2x},\ p_{2y},\ p_{2z}).$$

Any non-existing photon, in either input or output, is represented by the zero vector.

With the output being the reconstructed momentum vectors of the γ-rays, perhaps the most logical choice of loss function, based on the mean squared error introduced in eq. (3.3), is simply

$$\mathcal{L}\{\boldsymbol{p}_i\} = \sum_{i=1}^{m} \left\| \boldsymbol{p}_i - \hat{\boldsymbol{p}}_i \right\|^2, \qquad (4.3)$$

where $\hat{\boldsymbol{p}}_i$ is the corresponding correct vector. Since m is the same for each individual network, the factor $1/m$ is omitted in the python scripts to reduce the number of unnecessary floating point operations.

This error is still, as usual, averaged over the entire batch during training. Note that the use of cartesian coordinates handles the angular relationships automatically through the change of coordinates, also removing the need for periodicity in the loss function.

From testing with simpler networks, it was verified that this loss function does not cause any angular artefacts such as those discussed previously in section 4.3.1. Furthermore, this loss function features an affine gradient with respect to $\boldsymbol{p}$ and is therefore more computationally efficient to optimize.

4.3.3 Permutation loss

For multiplicites $m > 1$, each $\boldsymbol{p}_i$ has to be paired with its corresponding $\hat{\boldsymbol{p}}_i$. Since there is no inclination for the network to produce $(\boldsymbol{p},\ \boldsymbol{p}')$ over $(\boldsymbol{p}',\ \boldsymbol{p})$ for instance, these would result in two completely different values of $\mathcal{L}$. Olander *et al.* [3] solved this problem by calculating the loss function for all the $m!$ possible combinations and then selecting the one that has the minimal loss, i.e the least mean squared error. This could be done by having a script doing this assignment in a series of for-loops, but for backpropagation it would be necessary to write an additional gradient calculation.

The alternative is to use the default symbolic operations of Tensorflow. Let

$$Y = (\boldsymbol{y}_1,\ \boldsymbol{y}_2,\ \dots \boldsymbol{y}_n)^T \in \mathbb{R}^{n\times 3m}$$

be a output batch of size n and $\hat{Y}$ its corresponding labels. To calculate the mean loss as described above, consider the *permutation tensor* P of order 3 which comprises each of the $m!$ permutation matrices. With $m = 2$ for instance, the $2! = 2$, such matrices are the identity matrix $P_{12} = I$ and

$$P_{21} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Together with the *identity tensor* E, a tensor similar to P but only with $m!$ identity matrices, the error $\mathcal{E}$ for all possible combinations in the batch can be described in index notation as

$$\mathcal{E}_{ijk} = Y_{i\ell}E_{\ell jk} - \hat{Y}_{i\ell}P_{\ell jk},$$

where the last index in $\mathcal{E}$, E and P goes over the $m!$ permutations and identity matrices, respectively. The purpose of E is just to make copies of Y in order to pair them with every permutation of $\hat{Y}$. The squared error is then deduced from

$$\mathcal{L}_{ij} = \mathcal{E}_{ikj}\mathcal{E}_{mkn}\delta_{im}\delta_{jn},$$

which is a $R^{n\times m!}$-matrix containing the loss for every pairing. Here δ is the Kronecker delta defined as

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

The returned loss function is thus given

by minimizing these for each reconstruction and then taking the batch average, i.e

$$\mathcal{L} = \frac{1}{n} \sum_{j=1}^{n} \min_{i \leq m!} \mathcal{L}_{ij}.$$

This is the loss function used throughout this thesis.

4.4 METRICS OF PERFORMANCE

When analyzing the networks, a good way to represent the photon-reconstruction is through the "lightsaber" plot. Used by both previous projects [3, 4], they come in the form of three histograms, with axes showing the correct and reconstructed values of E, θ and ϕ on abscissa and ordinate, respectively. For a perfect reconstruction, the plot should show the identity represented by the blue diagonal lines. By utilizing the momentum as mentioned in section 4.3.2, we also introduce a new metric for evaluating network performance, the *Mean Momentum Error* (MME), as

$$\text{MME} = \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{p}_i - \hat{\boldsymbol{p}}_i \right\|, \tag{4.4}$$

where N is the number of data points. This is also simply referred to as the Mean Error, and serves as a good scalar metric of how well a network performs. Since this value treats all data points in $\mathbb{R}^3$ equally, it accounts for miss-identification of the existence of γ-rays by introducing an error either equal to the momentum of the reconstructed non-existent photon, or the non-reconstructed existing photon.

However, in order to quantify whether a network consistently succeeds at reconstructing existing photons as such, the events can be ordered into four categories Q_e, Q_i, Q_m, and Q_o[1], based on their correct and reconstructed energies as shown in table 4.1. As mentioned in section 4.2, the standard way of labeling a non-existent photon is by setting its energy and all its angles to zero. Since a network without classification neurons cannot explicitly label a photon as non-existent, a threshold ε is defined, with energies reconstructed below this threshold signifying non-existent photons. The value $\varepsilon = 0.01\,\text{MeV}$ is used, which is the lower limit for the energies generated in our data sets. Counting the number of events in each category gives additional information about the behavior of the network.

Table 4.1: *The different categories of γ-ray reconstructions. The labels Q_e and Q_o refers to those correctly registered as existing or non-existing, respectively, while Q_i and Q_m are the those invented or missed by the network.*

	$\hat{E} = 0$	$\hat{E} > 0$
$E > \varepsilon$	Q_i	Q_e
$E < \varepsilon$	Q_o	Q_m

Note how the number of events that end up in either one of these categories is highly dependent on the threshold. A Q_i photon could just as well be counted as a Q_o photon using a higher threshold. Still, those numbers provide an idea of how a certain alteration of a network changes its behavior. However, to more clearly visualize the workings of the network, the mappings of these categories are altered slightly compared to the previous reports [3, 4] to form a new type of plot shown in Fig. 4.2. Here, the Q_i events are displaced horizontally to a random point within the vertical bar, as to show the energy at which the γ-rays in this category are invented all while maintaining an appropriate z-axis for the plot..

[1] Those in the category Q_o are mapped close to the origin, hence the "o".

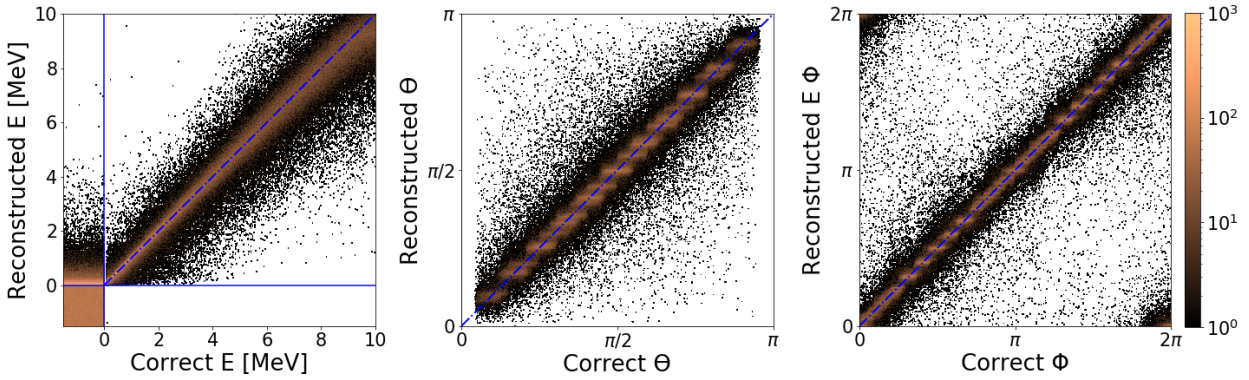


Figure 4.2: *Reconstruction of energy E, polar angle θ and azimuthal angle ϕ for γ-rays of maximum multiplicity $m = 2$, using a FCN of uniform width $N = 80$ and depth $D = 4$. The network is trained on a total of 1.5×10^6 events, including an additional 5×10^5 of fully empty events. Notice the lack of missing reconstructions in the Q_m area.*

4.5 FULLY CONNECTED NETWORKS

Perhaps the most elementary type of neural network is the FCN, making them a reasonable choice to investigate first. As described in section 3.1.1, a FCN of depth D consists of $D + 1$ Dense connections, each with a width of N_ℓ neurons. The FCN:s examined in this thesis are predominantly of uniform width $N_\ell = N$, $\forall \ell$. However, different network architectures with alternating widths are investigated in section 4.5.4.

Figure 4.3 shows a flowchart describing the general structure of the FCN. The input are the detected energies $\{E_i\}$ from the $N_0 = 162$ crystal detectors and the output are the m momentum vectors $\{\boldsymbol{p}_j\}$ of the reconstructed γ-rays. After each Dense connection a ReLU-function, given by eq. (3.2), is applied with the exception of the connection to the output. Here a linear activation is used for it be able to generate negative values, since ReLU ≥ 0, by definition.

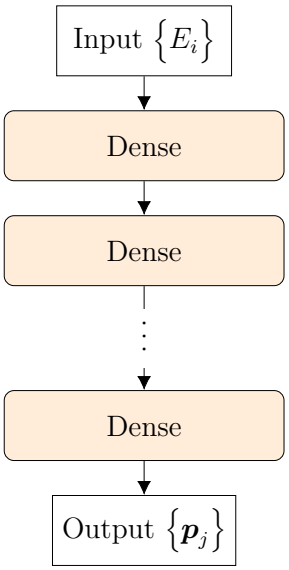


Figure 4.3: *Flowchart of a FCN with arbitrary depth connecting the input detector energies E_i with the reconstructed photon momenta $\boldsymbol{p}_j$. The coloured boxes represents the Dense connections between each of the layers, described in eq.* (3.1).

4.5.1 Maximum multiplicity of detected γ-rays

While the number of output neurons is fixed, it is still required that the ANN:s are able to train on any given maximum multiplicity of γ-rays. However this report is mostly considering the case $m = 2$.

Fig. 4.4 shows not only that the FCN:s are capable of this, but how well they perform with different m as well. A range of FCN:s of varying depths, all with a uniform width of 80, is trained on 1.5×10^6 simulated events with an additional 5×10^5 completely empty events. The case where $m = 2$ is also presented in Fig. 4.2. Higher maximum multiplicities give rise to more uncertain reconstructions as suggested by the ordering of the mean errors of different m in the figure. This is not surprising considering the calculation of the permutation loss function scales with $m!$, making the training of the network more difficult.

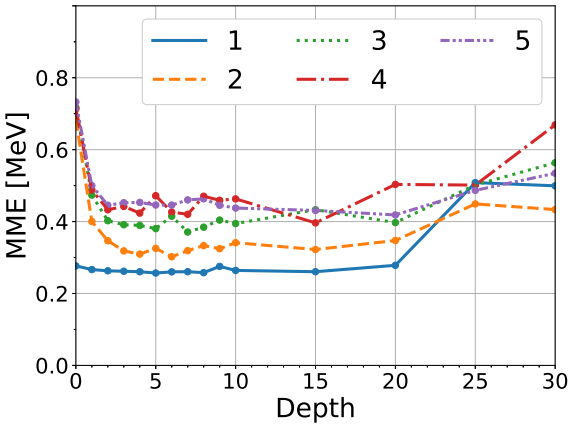


Figure 4.4: *Mean errors in reconstructed energies for models of depths between 0 and 30. Every hidden layer is of width 64. The models were trained on data sets of different maximum multiplicity, and are coloured accordingly.*

4.5.2 Utilizing batch normalization

To determine if batch normalization is beneficial to the networks performance, a series of FCN:s with or without utilizing batch normalization is compared to each other. The normalization is applied before the ReLU activations. Figure 4.5 shows a flowchart of these blocks of dense connections and batch normalizations.

Every network of uniform width 64 was trained on a maximum of 300 epochs and early stopping with a patience of 20 epochs. Figure 4.6 shows that using batch normalization in this case did not improve the performance of the FCN:s for depths below 20. The time for these to train did not differ significantly either. Those deeper than 20 seem to benefit from using the normalization but with an overall higher mean error than networks with lower depths.

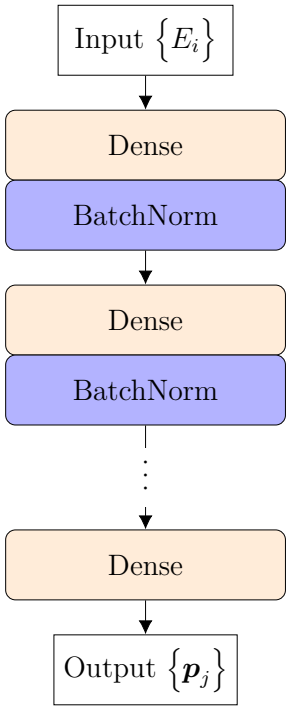


Figure 4.5: *FCN of arbitrary depth using batch normalization. Every hidden layer is of uniform width $N = 64$, save the last one connected to the output of width $3m$. Note that the activation function is applied at the end of each block.*

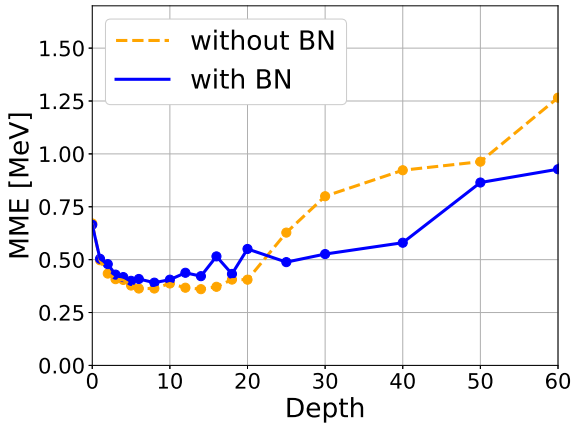


Figure 4.6: *Mean errors in reconstructed energies for models of depths between 0 and 60 with our without batch normalization. Every hidden layer is of width 64.*

4.5.3 Optimization of network dimensions

Every Dense connection is defined by a weight matrix $W^{(\ell)}$ and bias vector $\boldsymbol{b}^{(\ell)}$ as introduced in section 3.1.1. Between layer ℓ and $\ell+1$ there are $N_{\ell+1}(N_\ell+1)$ trainable parameters and for a network with a fixed width N throughout, the total number of parameters in a FCN with $D > 0$ is thus equal to

$$\begin{aligned} &N(N_0+1) \\ &+N(N+1)(D-1) \\ &+3m(N+1), \end{aligned}$$

where N_0 is the number of neurons in the input layer and $3m$ in the output respectively.

To deduce the optimal configuration of D and N in means of best performance for the least amount of parameters, a series of different FCN:s was trained systematically and then evaluated. With the mean error of the reconstructed momenta as the measurement of performance, a contour plot was created with Clough-Tocher interpolation implemented with the Python library *SciPy* [22]. This produced a piecewise cubic $\mathcal{C}^1$-surface shown in figure 4.7. Each network was trained with a maximum of 300 epochs and using early stopping with patience 5. Furthermore, a learning rate of 10^{-4} is used throughout the testing, as it was deemed optimal for similar networks in the same task. [4]

Fig. 4.7 shows that FCNs with depths larger than about 5 hidden layers do not improve the performance. In fact a depth $D = 4$ and width of about $N = 100$ neurons per layer seems to be the most favorable configuration considering its low number of parameters being about 5×10^4. This is marked with a red cross in the figure.

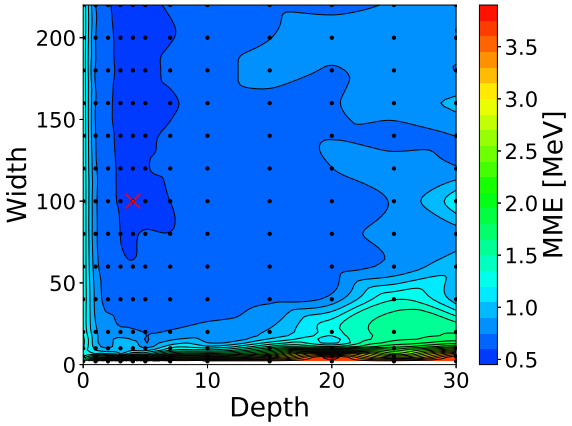


Figure 4.7: *Contour plot showing the interpolated mean errors for FCN:s with different depths and widths where each such configuration is marked with a dot. The red cross shows the selected FCN configuration used in the comparison with addback in the following chapter.*

4.5.4 Comparison of architectures

Until now, only FCNs of uniform width have been investigated. To see how these compare to non-uniform networks, three such FCNs of different architectures are analyzed. These are the *Triangle*, *Bottle* and *Inverted bottle* designs.

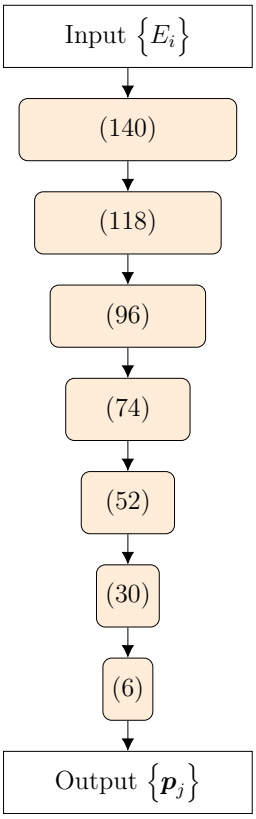


Figure 4.8: *The Triangle architecture with a depth* 6 *and maximum multiplicity* 2*. Each dense connection (coloured) is marked with the width N_ℓ of subsequent hidden layer.*

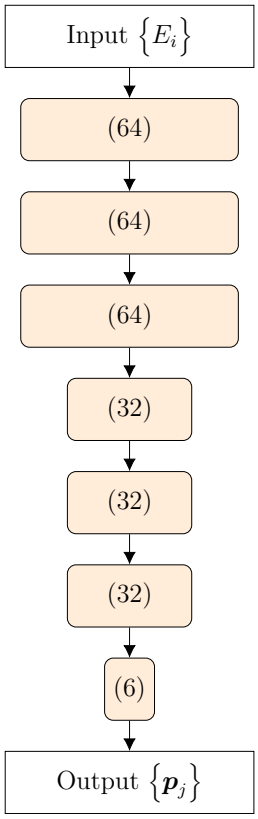


Figure 4.9: *The Bottle architecture with a depth* 6 *and maximum multiplicity* 2*. The Inverted bottle is the same, but with the two halves interchanged*

Firstly figure 4.8 shows the Triangle design with depth $D = 6$. The width N_ℓ decreases linearly between the input and the output layer as

$$N_\ell = N_0 - \frac{N_0 - 3m}{D + 1}\ell.$$

Every dense connection is activated by the ReLU-function, save the last one which yields the output through a linear activation.

The second design is shown in figure 4.9. It has two separate uniform parts, the first has a width 64 and the other narrower half has 32, hence the name Bottle. Because this has two parts of equal depths, this design must be of an even depth.

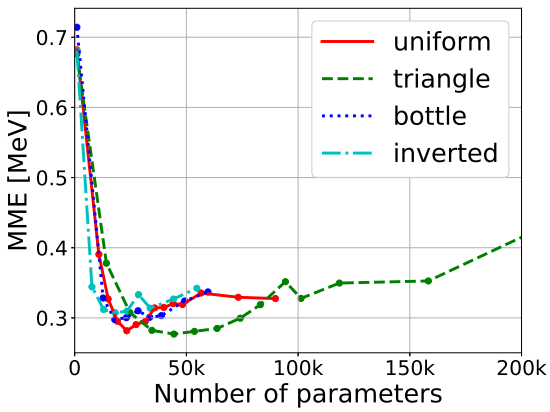


Figure 4.10: *Mean momentum error for the four different FCN architectures against the number of trainable parameters. Since fewer parameters are often much faster to train, this shows that there is no real reason not to use the uniform design.*

4.6 CONVOLUTIONAL NETWORKS

4.6.1 Rearranging the input

For a CNN to extract local information from neighboring crystals, the input data must first be reshaped in a way that accommodates for the geometry of the detector. Based on the work of Karlsson *et. al.* [4], this is done by applying a sparse binary transformation matrix that selects energies from specific crystals and arranges them into cluster arrays representing different areas of neighbouring crystals. The first element of each cluster array contains the energy of the central crystal, which is either an A or a D crystal, followed by the energies of its neighbours and second-neighbours. Note that in this context, a cluster refers to the collection containing the central crystal along with *all* of its neighbours and second neighbours — not only the subset of those registering a significant energy, as it is used in section 2.4. This results in clusters of size $S_A = 16$ for for each of the $N_A = 12$ A crystals and $S_D = 19$ for each of the $N_D = 30$ D crystals, which combined cover the entire Crystal Ball with some overlap. An example of a D cluster can be seen in Fig. 4.11.

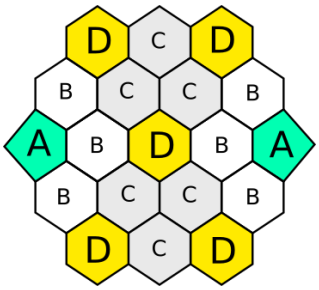


Figure 4.11: *Schematic representation of a D cluster. Note that the shapes and angular relationships of the crystals differ slightly from the real detector.*

Karlsson *et. al.* [4] chose to sort each cluster array by selecting the closest crystal to the beam exit as the first element among the neighbours and continue in counter-clockwise order (as seen from inside the detector). The same sorting was then made separately for the second-neighbors. This type of sorting, here abbreviated BE for *Beam Exit*-sorting, has problems due to occasional shifts among the indices of the second-neighbours and thus does not correctly take the local neighbourhood of the geometry into account. Due to this, we introduce a new sorting called CCT-sorting for *Consistent Crystal Type*.

For CCT, the cluster array centered around the crystal c can be written as $(c\ \mathbf{n}\ \mathbf{m})$ where $\mathbf{n}$ and $\mathbf{m}$ are arrays of the neighours and second-neighbours to c, respectively. The closest neighbours $\mathbf{n}$ are sorted counter-clockwise in the same way as for BE, with the exception of taking crystal shapes into account whenever applicable (i.e for D-crystals). In this case, the first neighbour element in the cluster array is the B-crystal closest to the beam exit. To sort the second-neighbours, we select the first element m_1 as the second-neighbour to c which is also neighbour to the first two entries of n. Mathematically, we do this by defining two sets $\mathcal{N}_1$ and $\mathcal{N}_2$ as the neighbours to the first and second element of $\mathbf{n}$, respectively. The first element of the second-neighbours can then be chosen as

$$m_1 = \{\mathcal{N}_1 \cap \mathcal{N}_2\} \setminus \{c\}.$$

After this, $\mathbf{m}$ is formed by continuing counter-clockwise, completing the cluster array $(c\ \mathbf{n}\ \mathbf{m})$.

For the pattern recognition of a CNN to be effective, one would prefer to not distinguish between the rotational symmetries of these clusters. For instance, a specific pattern produced by a photon when scattered between three C-crystals of a D-cluster should ideally be detected by the same kernel regardless of their position relative to the central D-crystal. A similar argument can be made for including mirror

images of the clusters, enabling a specific kernel to operate symmetrically on opposing clusters. To account for these symmetries, the cluster vectors are extended to include each rotated and reflected state of the cluster.

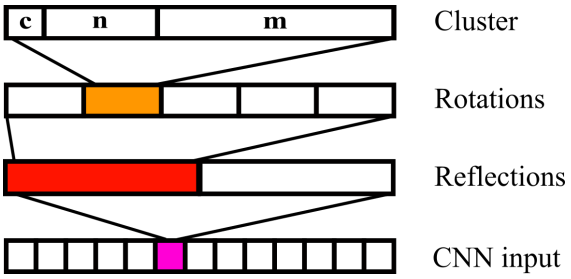


Figure 4.12: *Schematic representation of how the input vector to a branch of the CNN contains the rotations and reflections of a specific neighbourhood of crystals.*

Each rotated state features a copy of the previous, but with indices of the neighbours increased by one and second-neighbours by two (with wrapping), resulting in a counter-clockwise rotation. To handle reflections, the initial sorting of the neighbours is instead carried out clockwise, resulting in a reflection over the axis through the central crystal to the first neighbour. Incorporating these concepts in the transformation matrix generates the complete input to the CNN as shown in Fig. 4.12.

4.6.2 CNN models

Due to the different geometries of the A and D clusters, different kernel and stride lengths need to be applied for each central crystal type in order to summarize the information from the cluster. Therefore, the cluster vectors are fed to a sequence of parallel 1D-convolution layers. For each cluster type, the first convolutional layer has a stride and kernel length equal to the number of neighbours in the cluster, i.e $S_A = 16$ for the A clusters and $S_D = 19$ for the D crystals, summarizing the information of each cluster for a specific orientation. To handle the rotations and reflections, the last convolutional layers each have stride and kernel length equal to $r_A = 10$ for the A-clusters and $r_D = 12$ for the D-clusters. Since the kernel- and stride length of the convolutional layers are chosen with regard to the structure of the cluster vector, no padding is applied. By doing so, the output of these convolutional layers corresponds to the $N_A = 12$ A clusters and the $N_D = 30$ D clusters. At this point, the A- and D-branches of the network can be joined to a series of fully connected hidden layers, and finally connected to an output layer.

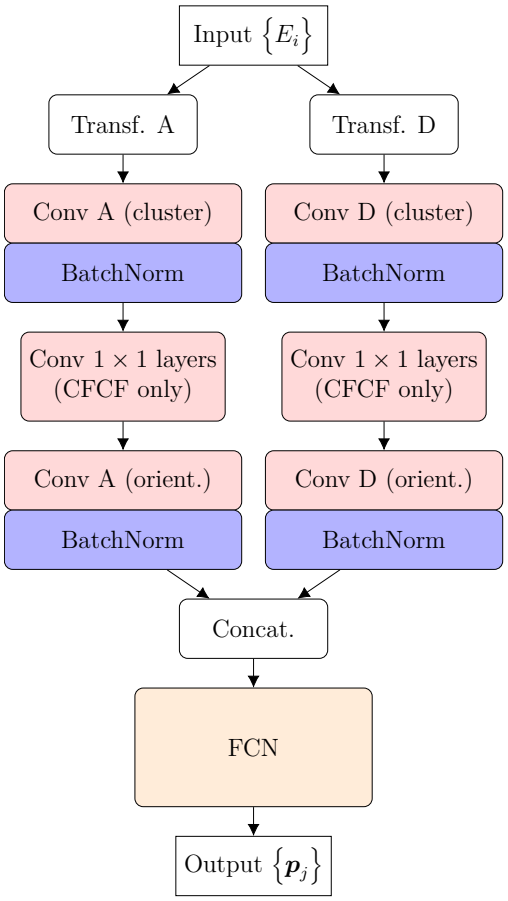


Figure 4.13: *Overview of the CCF and CFCF models with its two branches. This picture shows a version where Batch Normalization is applied. The Conv 1x1 layers are bypassed for the CCF model.*

Due to this structure, the model is referred to as the CCF-model (*Convolutional - Convolutional - Fully connected*) and is shown in Fig. 4.13. ReLU activations are applied to all convolutional and FC-layers, except for the last layer which has a linear activation.

The CCF model can be extended by introducing additional layers between the first and last convolutional layers. Specifically, a number of convolutional layers with kernel and stride of 1 are added, something that introduces more trainable parameters without reshaping the output. These layers are set to produce as many filters as their input layer, and are shown as "Conv 1x1" layers in Fig. 4.13. The introduction of these layers can be interpreted as "emulating" the transformations of fully connected layers for the feature maps, and therefore this is referred to as the CFCF model. In theory, this design should introduce greater headroom for learning patterns in the first convolutional layer. Furthermore, batch normalization can be added to both models after each convolutional layer.

4.6.3 Optimizing hyperparameters of the CNN

While the kernel and stride lengths of the CNN are given by the model as mentioned above, there are still many design considerations regarding the hyperparameters of the network. To simplify things, the learning rate is kept fixed at 10^{-4} and the number of feature maps f is kept constant between the corresponding layers of the different branches.

The initial tests consisted of optimizing the hidden layers of the CCF model. The tests were done with and without BN, training on data with a maximum multiplicity of 2 until halted by early stopping. The final hidden layers used a fixed width of 80. While this is wide in comparison to the number of output neurons, it is still not as wide as the concatenated output of the two branches given that 32 and 16 filters are used for the first and last convolutional layers, respectively. Training such models using a depth of 3, 5, 7 and 10 and evaluating them on a validation set yields the result shown in Fig. 4.14. From this figure, it becomes obvious that the CCF model does not benefit from BN, although it may be practical for higher multiplicities where the increased depth may result in a relatively lower MME. The figure also shows that this particular configuration of the networks does not improve with more hidden layers.

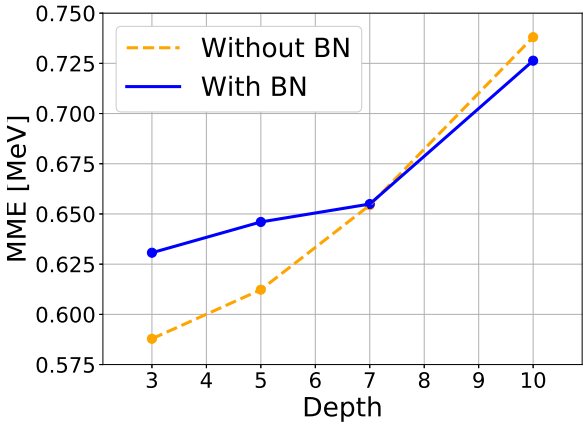


Figure 4.14: *Mean momentum error for CCF models with and without implementations of batch normalization.*

To investigate the effect of number of filters in the CCF model, the model is trained as before but with its FCN depth set to 3 with BN disabled. By compiling models with various numbers of filters at the first and last convolutional layer, and again training them until halted by early stopping, the models were tested using validation data. MME for the tested filter configurations are shown in Fig. 4.15, where one can see that filter configurations of more than 16 for each layer do not impact the performance of the network significantly. The smaller

configurations of 8 filters for the last layer, however, prove to be detrimental in terms of MME. Note that the models with more filters in the first layer tend to perform worse than those with an equal number in both layers.

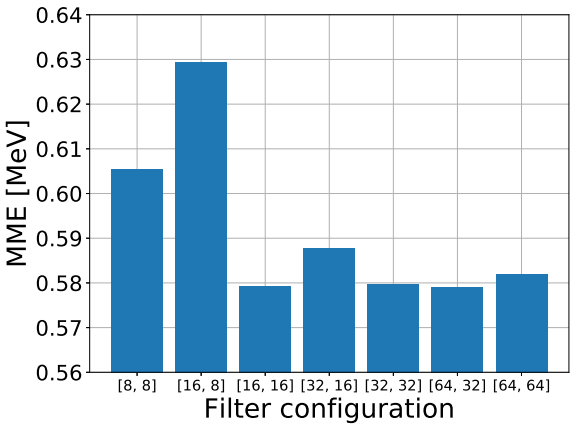


Figure 4.15: *Mean momentum error for different filter configurations of the CCF model. The numbers [a, b] represent the number of filters in the first and last layer, respectively.*

From this, one may draw the conclusion that 16 filters for each layer is the optimal configuration. However, with hopes of detecting greater multiplicities, the number of filters would likely have to increase. The same argument can be made for the depth of the network. To test this, four different CCF models were again trained using data with a maximum multiplicity of 5. These models consisted of all combinations of [16, 16], [32, 32] filter configurations and depths of 3 and 5, with other hyperparameters as before. Here, the most expensive network to train in terms of number of parameters, i.e [32, 32] filters, $D = 5$ turned out to yield an MME 0.041 MeV smaller than the cheapest, while having around twice as many trainable parameters.

To compare the CFCF model to the CCF, a number of models are trained on data with a maximum multiplicity of 2. The models use 32 convolutional filters per layer and 3 hidden layers with a width of 80. Each CFCF features 1-10 intermediate convolutional layers. The validation MME is shown in Fig. 4.16. Comparing the result to the 32 filter configuration of the CCF from before, one can see a slight improvement in MME when using 2–3 intermediate layers. However, the absence of a trend in this plot suggests that the training of these networks is limited by the learning rate.

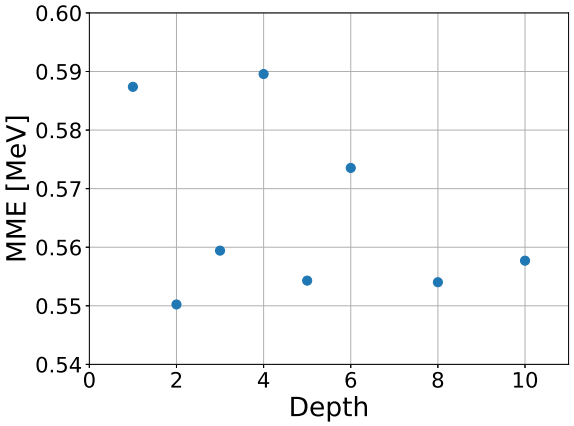


Figure 4.16: *Mean momentum error for the CFCF model when using different numbers of intermediate layers.*

4.6.4 Dealing with existence

Based on the category system introduced in Table 4.1, the events reconstructed by the CNN has in the aforementioned tests very nearly consisted of 75% Q_e photons and 25% Q_i photons for data with maximum multiplicity of $m = 2$. For $m = 5$, these numbers were very close to 60% Q_e and 40% Q_i. These numbers seem fitting, as for instance in the case of $m = 2$, the data contained a 1:1-ratio of single and double γ-ray events. The reason for other categories remaining (almost) empty is that the network rarely reconstruct low enough energies to fall below a threshold of $\varepsilon = 0.01$, let alone without being presented empty events (no registered energy, all labels set to zero).

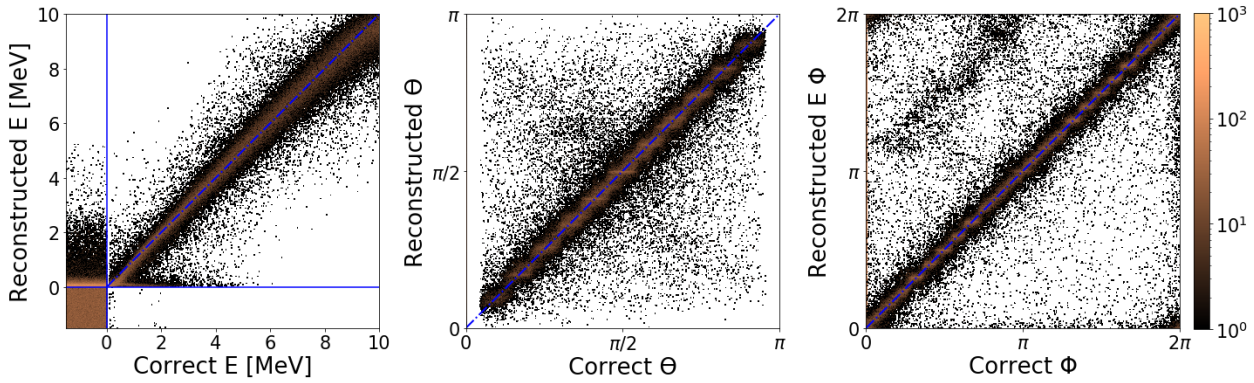


Figure 4.17: *Histogram over the reconstructions of the CFCF model trained on data with $m = 2$ including empty events. This particular model uses [16, 16] filters, a depth of $D = 3$ and 3 intermediate convolutional layers.*

By training with empty events, a more accurate reconstruction was hoped to be achieved; both in terms of MME and that few γ-rays are missed or invented by the network (false positives). This goal can only somewhat be formalized by finding a network such that the Q_i events are of lower energy. To observe this, several CNN models were trained on data with $m = 2$ and an additional $5\times^5$ empty events. The best in terms of MME was a CFCF model with 16 filters in each layer, 2 intermediate layers and 3 hidden layers. Its reconstruction, shown in Fig. 4.17, yielded an MME of 0.296 MeV.

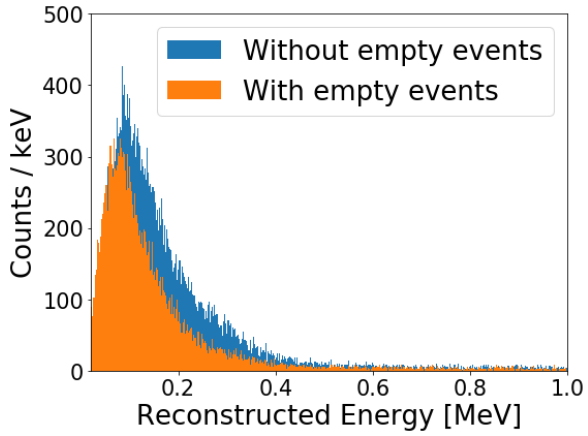


Figure 4.18: *Histogram comparsion between the Q_i events of CFCF models trained with and without empty events.*

Comparing this to earlier trained models, it is clear that the MME is greatly impacted by training with empty events.

From retraining the same model on data not containing empty events and comparing the distribution of Q_i events, as seen in Fig. 4.18, it is clear that it also improves the network in this regard.

4.7 GRAPH NETWORKS

The motivation to use the graph $\mathcal{G}$ of the neighbouring detector crystals is understood through the aggregation (3.4) and its relation to the addback routines as described in section 2.4. Using this in the beginning of a network integrates the detected mean energy in each neighbourhood of the input neurons (i.e. the crystal detectors), just as addback does, into a fully connected network. Without this the network does not know how the neurons are related to each other and has to learn this during training. Since $\mathcal{G}$ is known, the construction of such a GNN is rather straightforward.

First, the adjacency matrix A, which encodes most of the information regarding $\mathcal{G}$, is constructed. It is a symmetric

162×162-matrix where

$$A_{ij} = \begin{cases} 1, & \text{if crystals } i,\ j \text{ are neighbours,} \\ 0, & \text{otherwise.} \end{cases}$$

As mentioned in section 3.3, the matrix should include self loops as well. This is done by adding the identity matrix, i.e. $\hat{A} = A + I$. The diagonal node degree matrix D is simply the row-wise sum of $\hat{A}$ in its diagonal.

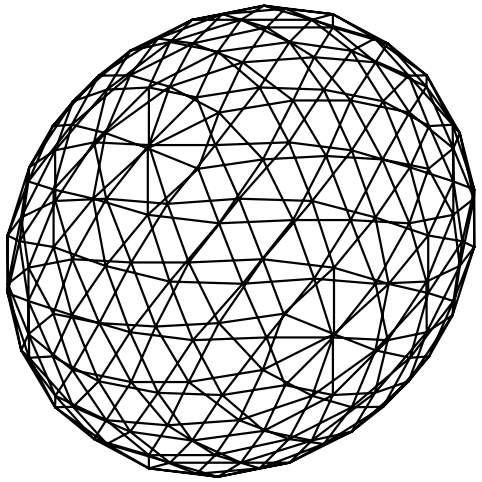


Figure 4.19: *The graph $\mathcal{G}$ representing the relationships of every crystal detector in the Crystal Ball. Each node is mapped to the surface center of corresponding crystal.*

A GraphDense connection is applied directly on the input layer and preserves the width N_0 to the following layer. Being very similar to a Dense connection with the same kind of trainable parameters W and $\boldsymbol{b}$, the number of trainable parameters is given by $N_0(N_0+1) = 26\ 406$ in the case of the Crystal Ball detector.

Testing different GNN:s can be done in multiple ways. The selected approach is to compare 3 such networks with the uniform FCN, which was investigated in section 4.5. The first network GNN1 is illustrated in figure 4.20, consisting of a single graph convolution and a subsequent FCN.

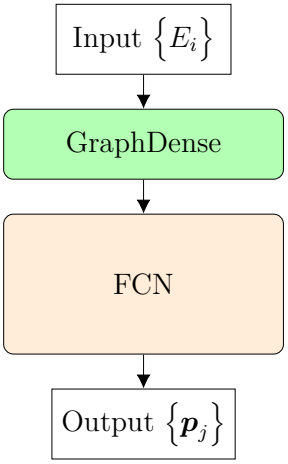


Figure 4.20: *The GNN1 model with a single GraphDense connected to a consecutive FCN. GNN2 is similar in design with one additional GraphDense directly after the first one.*

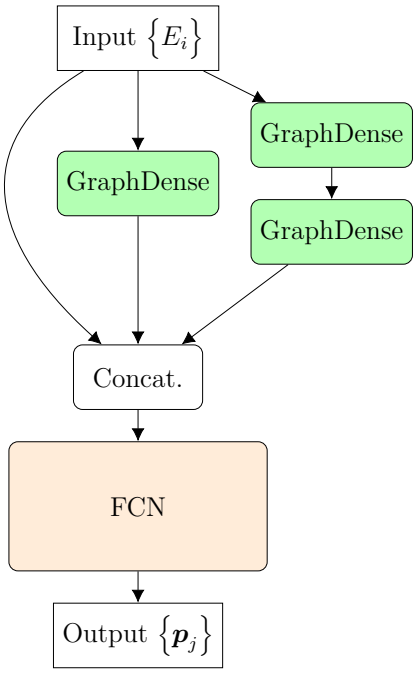


Figure 4.21: *The GNN3 design with its three branches which are concatenated to a $N = 3 \times 162$ wide hidden layer. This is followed by a FCN before the output layer.*

The second, called GNN2, is the same as GNN1 with an additional convolution directly after the first one. While GNN1 is feeding the neighbourhood mean energy for

each node, GNN2 deals with the more extensive area including second neighbours as well.

The last one is unsurprisingly named GNN3 and stands out from the other two with three branches between the input neurons and the following FCN. This covers both the smaller neighbourhood as GNN1 does and likewise the larger one as the second design.

The third branch in GNN3 is supposed to further include the individual energy deposits as well. As shown in figure 4.21, these three branches are concatenated feeding the FCN with a 3×162 wide input layer.

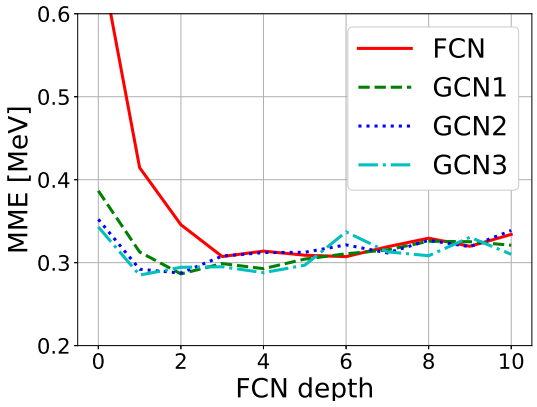


Figure 4.22: *The mean momentum error in reconstructions made with a FCN and the three types of GNN:s with a varying depth in the subsequent FCN:s shown in figures 4.20 and 4.21. Each network was trained on data with $m = 2$.*

Varying the depth of the FCN:s, these four network designs were trained on data of maximum multiplicity 2. The mean momentum error in the event reconstruction for each network is shown in figure 4.22.

One should note that a GNN of zero FCN depth still has these GraphDense connections and therefore have more trainable parameters than the corresponding uniform FCN. With this in mind, there is not a significant difference in MME between the four networks at higher depths. Nevertheless the GNN:s with a number of subsequent dense connections seems to perform slightly better than the FCN at any depth.

4.8 BINARY CLASSIFICATION

The ANNs examined so far are trained to model a continuous spectrum, in this case the γ-ray momenta $\{\boldsymbol{p}_i\}$. Nevertheless, this method has the limitation of not being able to explicitly imply whether a photon is really there or not in the reconstruction.

To introduce a binary classification neuron for each event is nothing new. Karlsson *et al.* [4] tried adding some different contributions to $\mathcal{L}$ by training with an additional neuron $\{\mu_i\}$, one for every γ-ray. Their approach was to modify $\mathcal{L}$ in eq. (4.1) in a variety of different ways to include these classification neurons. The loss function would in other words depend on four variables per hit as $\mathcal{L}\{E_i, \theta_i, \phi_i, \mu_i\}$. During training the correct $\hat{\mu}_i \in \{0,\ 1\}$ was set to equal zero for non-existent γ-rays and vice versa. They concluded that these networks was in fact able to classify missing photons to some extent, but performed even worse in reconstructing the existing in comparison to the pure regression networks.

Having the completely refurbished permutation loss function (4.3) this opens up a myriad of new possible ways to tackle this problem. One idea is to add an extra term to $\mathcal{L}$, proportional to the binary cross-entropy $H(\beta_i)$ as introduced in section 3.1.2, where β_i represents any floating-point number between 0 and 1. It is the network's best guess whether the γ-ray exists or not. This classification neuron is analogous to μ_i but another name is used to avoid any confusion with that method.

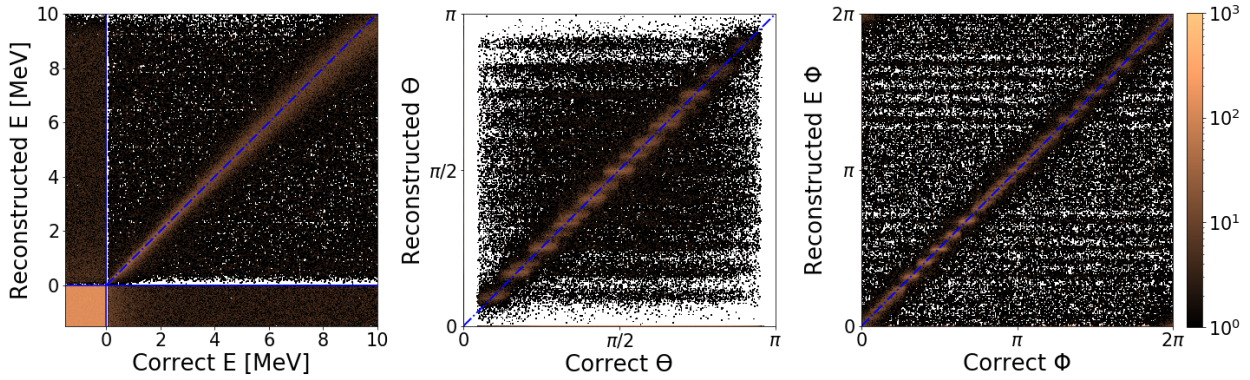


Figure 4.23: *Reconstruction of energy E, polar angle θ and azimuthal angle ϕ for γ-rays, using the binary classification model. The FCN is of uniform width $N = 80$ and depth $D = 4$ and was trained on data with maximum multiplicity $m = 2$ on a total of 1.5×10^6 events, including an additional 5×10^5 of fully empty events. The line artefacts in the two angle plots is manifestations of the 162 detector positions.*

The loss function $\mathcal{L}\{\boldsymbol{p}_i,\ \beta_i\}$ is given by

$$\mathcal{L} = \sum_{i=1}^{m} \left[\left\| \hat{\boldsymbol{p}}_i - \boldsymbol{p}_i \right\|^2 + \lambda H(\beta_i) \right] \qquad (4.5)$$

The coefficient λ is used to decide how much this binary classification may affect $\mathcal{L}$ and is here simply set to one. Further investigation will be needed to obtain a more suitable value.

A problem arises when adding these extra classification neurons relating to the network design. As before, the reconstruction of $\{\boldsymbol{p}_i\}$ must still be treated as a regression problem and the last activation producing these values needs to be linear. Since $\{\beta_i\}$ is a probability in the interval $(0,\ 1)$ its activation function needs to constrain the output accordingly. The sigmoid function does just this. Therefore, the output activations are separated as illustrated in Fig. 4.24. Before calculating $\mathcal{L}$, every classification neuron is paired with a reconstructed $\boldsymbol{p}_i$. The minimum permutation loss is then acquired as described in section 4.3.3 but now with 4 variables instead of 3

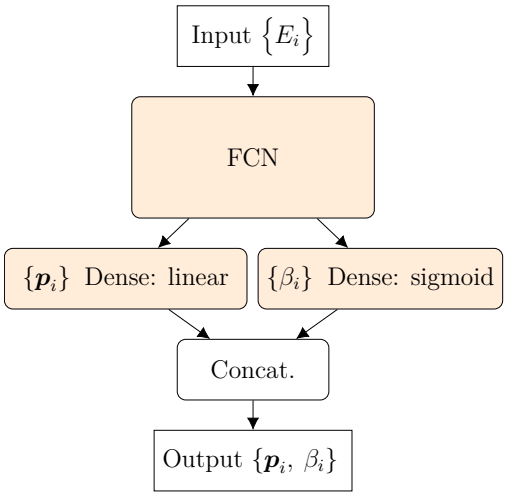


Figure 4.24: *The binary classification model with the two types of activations shown. These are concatenated together before the output.*

for each event being $(p_x,\ p_y,\ p_z;\ \beta)$.

This method of using the loss function (4.5) was not examined thoroughly enough to be able conclude whether the approach of implementing an extra term for the binary classification has any potential or not. A simple test was made using an FCN with $N = 80$ and $D = 4$, which has shown to per-

form rather sufficiently with the ordinary momentum loss (4.3). The resulting reconstruction is illustrated in Fig. 4.23, where $\lambda = 1$ is used.

The left plot shows the reconstructed energies against the correct ones, and suggests that while a large amount of empty events are correctly registered, i.e. in Q_o, the overall reconstruction is substandard.

Compared to the networks examined in section 4.5.3 with a mean momentum error usually lower than 1 MeV, this classification network has a MME of about 1.7 MeV. Considering that many of these events are registered in Q_o, meaning they do not contribute to this metric, this network is profoundly worse than most of those explored in this thesis. Furthermore, a fair amount of events are registered in Q_m and thus falsely omitted from the event reconstruction.

Perhaps another configuration of the proportionality coefficient λ would result in more acceptable reconstructions. Thus further investigations is required. The use of terms other than the binary cross entropy in (4.5) could improve this method as well. For further discussion, see section 6.4.

Chapter 5

Results

To assess how well the ANNs performed in reconstructing detector data, the networks were compared to the first-neighbour addback routine described in sec. 2.4. This was done by having them reconstruct a more realistic evaluation set that involves a Doppler shifted signal of some fixed energy and background radiation. This method of analysis and comparison was inherited from [3].

5.1 METHODS OF COMPARISON

The previous chapters outline a comprehensive investigation with the goal of determining the optimal design for an ANN for multiplicities equal to or lower than two. Three such networks selected to be compared with addback is described below.

The first model is a FCN with 4 hidden layers, each with 80 neurons. This configuration of depth and width is motivated by the work in section 4.5, specifically by the results presented in Fig. 4.7. This network yielded a low MME considering it only has 32 966 trainable parameters, the lowest amount among these three networks.

The second model is a CNN, based on the CFCF model detailed in section 4.6.4. This network contains a total of 125 574 trainable parameters.

The last model is the GNN with three branches and a total of three GraphDense connections illustrated in Fig. 4.21. Its subsequent FCN includes a single hidden layer of width 80, resulting in a network with a total of 118 178 parameters.

Each network was trained on a data set containing a uniform distribution of energies in the range $0.01 - 10\,\mathrm{MeV}$ with a maximum multiplicity of $m = 2$, including empty events, as shown in Fig. 4.1. After this, they were evaluated on a second more realistic data set. In this set, each event has a 50 % chance of containing a γ-ray from background radiation in the range $0.01 - 1\,\mathrm{MeV}$, and a 10 % chance of containing a Doppler shifted signal γ-ray with a fixed energy $\hat{E}$ in the beam frame of reference. The speed of the latter is uniformly distributed between $\beta = 0.6 - 0.7\,\mathrm{c}$. Since the probabilities for each type of γ-ray to occur are independent, the validation set also has a maximum multiplicity of $m = 2$. Due to the relativistic effects discussed in section 2.3, the reconstructed energies are transformed to beam frame of reference using eq. (2.1). The outputs of the network are then compared with addback, as shown in the following section 5.2.

To each output, whether from addback or a neural network, the superposition of a Gaussian and an exponential curve is fitted. The Gaussian curve, labeled f_s, represents the signal, whereas the exponen-

tial curve, f_b, represents the background radiation. Serving as a metric of how well the signal can be distinguished from the background, the signal-to-background ratio R is calculated as

$$R \equiv \frac{\int f_s(E)\, dE}{\int f_b(E)\, dE}, \tag{5.1}$$

where the integrals are evaluated on the interval $\mu \pm \sigma$, with μ and σ being the mean value and standard deviation of the Gaussian fit, respectively. The R-values obtained for each network can be seen in Fig. 5.4.

Finally, the absolute difference between the reconstructed and the correct energy is calculated for each network. Here, the reconstructed energy E is is the mean value μ of the signal fit f_s. This is done for correct signal energies $\hat{E}$ in the range $1.5 - 8.0$ MeV. This is presented in Fig. 5.5.

5.2 COMPARISON WITH ADDBACK

To verify the effects of a momentum-based Cartesian loss function, we consider the problem of reconstructing Doppler shifted γ-rays with energy 3.5 MeV, as investigated previously by Karlsson *et al.* [4]. The preceding study utilized a modulus-based loss function which, despite yielding accurate reconstructions for high energies, gave rise to artefacts in the reconstruction of lower energies. The equivalent task using momentum loss can be seen in figure 5.1, where the blue line represents our reconstruction using a FCN, and the red line represents reconstructions made by addback. This image clearly portraits the eradication of the previously mentioned artefacts, which took the shape of an additional peak in the region of lower energies.

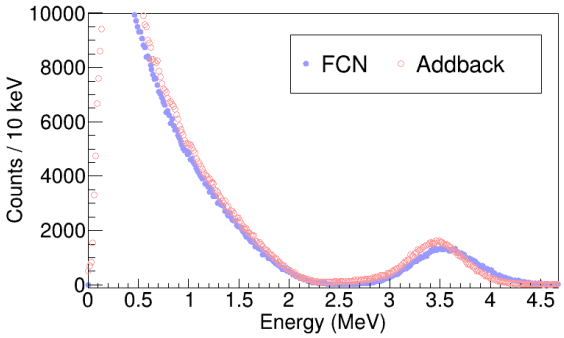


Figure 5.1: *Beam frame energy spectrum reconstructed by addback shown in red and the FCN-network in blue. The Doppler shifted γ-ray signal has an energy of* 3.5 MeV.

Eliminating these artefacts allows for the reconstruction of low energy events. The FCN-reconstruction of such an event, at 1.5 MeV, is represented in Fig. 5.2. Here, the fitted signal- and background curves can be seen as the dashed and dotted lines, respectively. Moreover, the reconstructed energies for each event are represented in table 5.1, which further demonstrates the networks' ability to reconstruct low-energy γ-rays. Figure 5.3 represents a zoomed in version of the 3.5 MeV event, shown in figure 5.1, as to give a visual comparison to the equivalent image of the preceding study.

Table 5.1: *Reconstructed signal energies of addback and the three networks. The energies are in* [MeV].

Correct	Addback	FCN	CNN	GNN
1.5	1.48	1.50	1.54	1.53
2.0	1.94	2.00	2.02	2.01
2.5	2.49	2.57	2.59	2.58
3.5	3.46	3.57	3.59	3.56
5.0	4.90	5.09	5.22	5.13
6.0	5.88	5.74	5.82	5.79
7.5	7.33	7.34	7.47	6.92
8.0	7.82	7.79	7.86	7.38

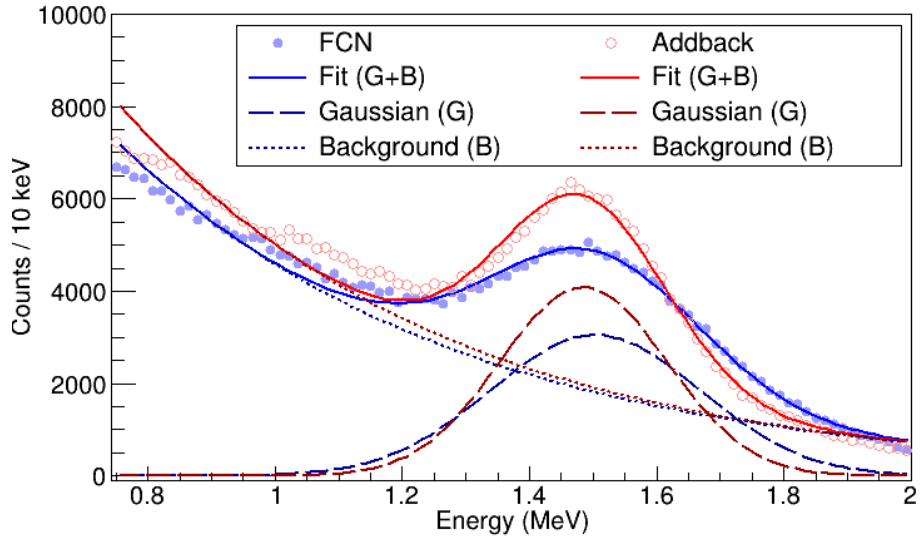


Figure 5.2: *The energy spectrum in the beam frame of reference reconstructed by addback shown in red and the FCN in blue. A Gaussian curve f_s, as well as a exponential f_b, is fitted for the the Doppler shifted γ-ray signal of energy of* 1.5 MeV *and the background radiation respectively.*

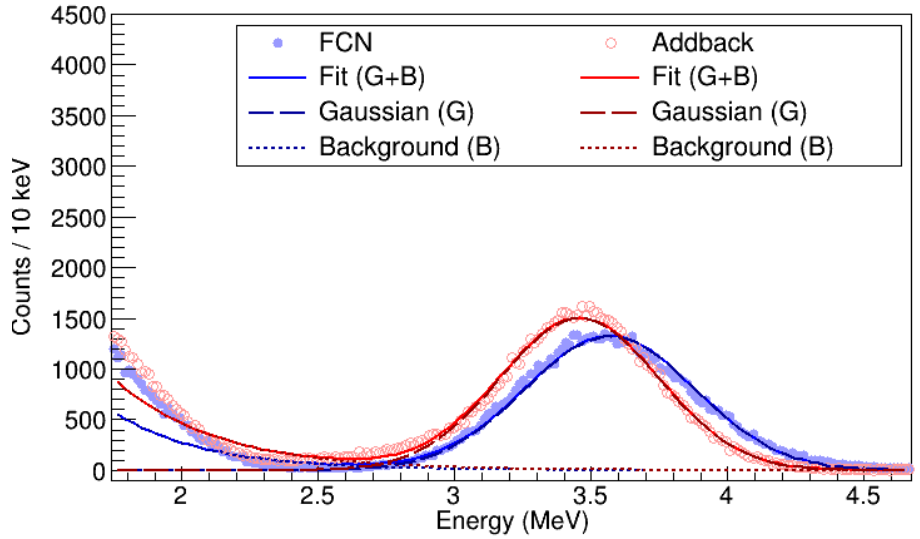


Figure 5.3: *The energy spectrum in the beam frame of reference reconstructed by addback shown in red and the FCN in blue. The Doppler shifted γ-ray signal has energy of* 3.5 MeV*.*

The signal to background ratio, as seen in Fig. 5.4, implies similar reconstruction performance between addback and the trained neural networks, entailing the networks have little difficulty reconstructing any of the investigated energies. The best R-value for the ANNs is reached between 3–6 MeV, surpassing that of addback, before decreasing at higher energies.

The mean and standard deviation of the reconstruction errors for the ANNs and addback are illustrated in Fig. 5.5. Each error is calculated from evaluations of 1.5 to 8.0 MeV, using the data represented in table 5.1. The FCN and CNN evaluations resulted in a mean and standard deviation which show similar, yet slightly worse, performance than addback. The GNN performed significantly worse in this metric.

The best performance among the ANNs is shown by the CNN network, which resulted in mean error of $1.01 \cdot 10^{-1}$, as compared to the mean error of addback being $8.75 \cdot 10^{-2}$, both with a standard deviation of $\sim 6.2 \cdot 10^{-}2$. However, in relative difference of the energies, it is clear that the FCN produced the most accurate predictions across the energy range. Though resulting in higher mean errors than previous studies, each network performed exceptionally well in terms of consistency over a large range of energies.

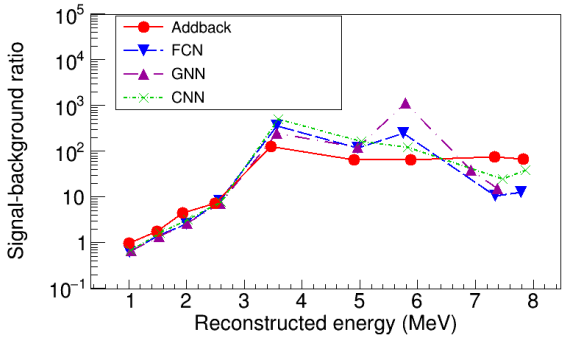


Figure 5.4: *Signal to background ratio, R, for evaluation of energies ranging from* 1.0–8.0 MeV *in beam frame. The marker style of each method is illustrated in the legend.*

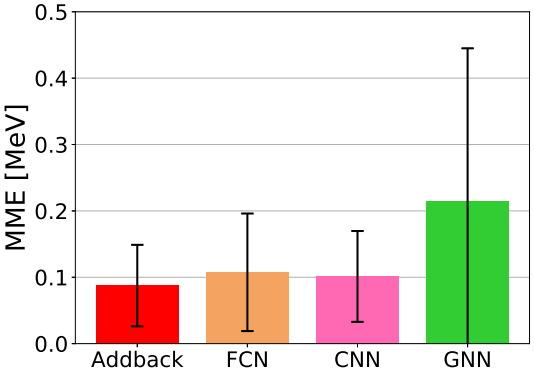


Figure 5.5: *Illustration of the absolute difference between the reconstructed energies E and the correct energies $\hat{E}$ listed in table 5.1. The hight of each box represents the mean error, and the lines represent the standard deviations.*

Chapter 6

Discussion

With the aim of this thesis being to examine methods using ANNs for the task of event reconstruction of γ-rays, progress in further developing such methods have been made on the foundation of previous work by [3, 4]. This chapter discusses these improvements, other less significant results, and suggestions for further investigations.

6.1 CONCLUSION

In contrast to the results of [4], the reconstructions of the energy spectrum presented in section 5.2 does not introduce any artefacts such as peaks in the background radiation region. In fact, these are sometimes even hard to distinguish from those of addback as is the case in figure 5.1. However, the closer the energy of the signal γ-rays are to that of the upper or lower limits of the distribution, the less accurate the reconstruction is. This is not surprising or even necessarily bad. If one wishes to perform measurements at other energy ranges, the simple solution would be to train on energies accordingly.

As shown in Fig. 5.5, the FCN seems to produce an overall more accurate reconstruction than the other two networks. Considering this has about 27 % fewer trainable parameters, the FCN using the permutation loss function is the most appealing method. The CNN and GNN do not improve upon this simpler design, even though they were more planned out and maybe even overengineered.

From optimizing the hyperparameters of the networks trained in this thesis, a recurring pattern has been that the deeper networks performed worse than those with fewer hidden layers. With our approach of testing, it may mean that the final designs end up being approximations of addback, simply weighing in the energies of different crystals together. This could be bad, since the main incentive to using neural networks is the chance of it finding more complex relationships within the data, not otherwise easily modeled by the means of conventional algorithms. To do this, while still facilitating the option for learning simpler patters, one could implement a model based on the ResNet or ResNext architechtures [23, 24]. Such models feature a data flow which can bypass several layers, which would in theory enable deeper network structures while simultaneously having shorter forward-propagation connections for learning simpler relationships. For the same reasons, alternative methods utilizing parallel layers with, or combined through, max pooling could also see success.

Perhaps the most decisive feat of this thesis is the momentum loss function (4.3) being the squared error in the γ-ray momentum vector. Not only is it based on the standard choice of loss for this type of regression problem, as mentioned i section 3.1.2, it also eliminates the need of relative weighting factor such as those in eq. (4.1) and (4.2).

Using batch normalization does not seem to do much for the networks, if not making them perform worse. This is not so problematic in this case, considering that the networks investigated are not so deep, making the vanishing gradient problem, as discussed in sec. 3.4, barely noticeable. However, it may still see use in the context of deeper networks or when using a classification node.

6.2 CONVOLUTIONAL NETWORKS

As briefly mentioned in sec. 4.6.3, some results indicate that the networks may have underperformed due to their training. Even though the networks were trained on extensive data sets, using early stopping to ensure a balance of validation and training data loss, the use of a fixed learning rate can lead to some parameters "overshooting" their local optima. One way to address this is to, once halted by early stopping, reinitialize the training again with a lower learning rate. This would likely also increase the performance of the FCN and GNN.

Furthermore, the optimization of the hyperparameters of the CNN (see sec. 4.6.3) were mostly done using simulated data only containing one or two γ-rays, i.e. no empty events. While the process itself most certainly led to a more refined model, it is possible some choices of hyperparameters would be different if the networks were trained on data more similar to that of the final evaluation.

6.3 GRAPH NETWORKS

While the GNN did not improve upon the reconstruction of the FCN, there might be potential with this method when scaling up the number of inputs. It would be interesting to see if encoding the graph into the network a priori could reduce the number of trainable parameters needed when dealing with much larger detectors, where also more prevalent Compton scattering lead to more crystal detectors registering a single initial γ-ray. Perhaps the Crystal Ball has a too low solid angle resolution for GNNs to be an appropriate method for this task. Further investigations of other aggregation rules as discussed in 3.3 would then be more interesting to make. In [11] a variety of alternatives to the spectral rule (3.4) is mentioned, where some even make use of convolutions.

6.4 CLASSIFICATION NETWORKS

The use of additional classification neurons has been speculated in [3, 4] to give the network the ability to better determine the existence of an event. While more events are correctly registered as empty events, the accuracy in the reconstruction of existing γ-rays have so far only been shown to decrease. This result was documented in [4] and yet again observed in Fig. 4.23.

Further investigation should be focused on the design of the loss function, since this seems to have the largest impact in the performance of a neural network. A few suggestions is firstly to optimize the proportionality coefficient λ in eq. (4.5). Using another logistic loss function instead of binary cross entropy could perhaps be a good idea as well. An example is the *Hinge loss* defined as

$$\mathcal{L} = \max\left(0, 1 - \beta\hat{\beta}\right),$$

where $\hat{\beta} \in \{-1, 1\}$ is the correct binary la-

bel and β the predicted value in an appropriate domain. For instance when $\beta\hat{\beta} < 1$, the classification is either wrong ($\beta\hat{\beta} < 0$) or correct but with a small confidence ($0 \leq \beta\hat{\beta} < 1$), meaning that Hinge loss also penalizes inconfident predictions.

Furthermore, one could get rid of the classification term in the loss function and try to approach the problem in another way. On suggestion is the slightly modified

$$\mathcal{L} = \sum_{i}^{m} \left[\beta_i \left\| \boldsymbol{p}_i - \hat{\boldsymbol{p}}_i \right\|^2 + (1 - \beta_i) \left\| \hat{\boldsymbol{p}}_i \right\|^2 \right],$$

with β_i being sigmoid activated. In this way, no ambigous factor λ is needed.

6.5 METHODS OF COMPARISON

An important aspect of all scientific studies is to find accurate methods for evaluating the final result. The primary methods used in this study where the signal to background ratio, R, and the absolute difference between the correct and the reconstructed energies. Both methods where adapted from previous theses, with little to no change, in order to give a fair comparison to previous studies. However, after some investigation, we found that the evaluation of the R-value is greatly affected by the choice of the parameters used to calculate a start approximation to the curves. Simply changing the initial angle of the exponential fit by a factor 0.5 changed the final R-value by a factor 10^4, implying the approximation is highly unstable.

Bibliography

[1] R. S. Simon, "The darmstadt-heidelberg crystal ball," *Journal de Physique Colloques*, vol. 41, 1980.

[2] S. Lindberg, "Optimised use of detector systems for relativistic radioactive beams," Master's thesis, Inst. för fysik, CTH, Göteborg, 2013.

[3] J. Olander, M. Skarin, P. Svensson, and J. Wadman, "Rekonstruktion från detektordata med hjälp av neurala nätverk: En studie i tvärsnittet mellan kärnfysik och maskininlärning," Tech. Rep., 2018.

[4] R. Karlsson, J. Jönsson, M. Lidén, and R. Martin, "Event reconstruction of γ-rays using neural networks: Going deeper with machine learning into the analysis of detector data," Tech. Rep., 2019.

[5] A. Knyazeva, J. Parka, P. Golubeva, and J. Cederkäll, "Properties of the csi(tl) detector elements of the califa detector," *Nuclear Instruments Methods in Physics Research*, vol. 940, pp. 393–404, October 2019.

[6] W. R. Leo, *Techniques for nuclear and particle physics experiments: a how to approach*. Springer, 1987.

[7] W. Rindler, *Relativity: Special, General and Cosmological*, 2nd ed. Oxford Press, 2006.

[8] V. Panin, "Fully exclusive measurements of quasi-free single-nucleon knockout reactions in inverse kinematics," Ph.D. dissertation, Technische Universität, Darmstadt, 2012. [Online]. Available: https://tuprints.ulb.tu-darmstadt.de/id/eprint/3170

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[10] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *ICLR conference paper*, 2015.

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE TNN*, vol. 20, 2009.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," Presented at the ICLR, Tech. Rep., 2017.

[13] T. S. Jepsen. (2019) How to do deep learning on graphs with graph convolutional

networks. [Online]. Available: https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-62acf5b143d0

[14] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_3

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," Presented at the International Conference on Machine Learning, Tech. Rep., 2015.

[16] ONEIROS. (2020) Keras: The python deep learning library. [Online]. Available: https://keras.io/

[17] G. B. Team. (2020) Tensorflow: Essential documentation. [Online]. Available: https://www.tensorflow.org/guide/

[18] HPC2N. (2016) Kebnekaise. [Online]. Available: https://www.hpc2n.umu.se/resources/hardware/kebnekaise

[19] Geant4 Collaboration CERN. (2020) Geant4. [Online]. Available: http://www.geant4.org/geant4/

[20] H. T. Johansson, "Ggland – command line simulations," *GSI*, vol. 2017-1, 2013.

[21] *ROOT Data Analysis Framework – Users Guide*, May 2018. [Online]. Available: https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuideA4.pdf

[22] T. S. Community. (2019) Clough-tocher interpolation. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CloughTocher2DInterpolator.html#scipy.interpolate.CloughTocher2DInterpolator

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[24] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.