Studies on using spherical data in deep neural networks for the discrimination of electron and positron in JUNO

von
Gerrit Leon Roellinghoff

Bachelorarbeit in Physik

vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen

am 23 Juli 2018

angefertigt am
III. Physikalischen Institut B der RWTH Aachen

bei
Prof. Dr. Christopher Wiebusch
1 Introduction

Neutrinos are particles that only interact via weak force and gravity, and are, out of all known elementary particles, the particles with the smallest mass. In fact their mass is so small, that it was long thought to be zero. Now we know that the neutrino is not only not massless, but that neutrinos are a quantum superposition of three mass eigenstates with different, albeit tiny, values. Due to the slight differences in mass for the eigenstates, the phases of these eigenstates advance at different rates, leading to the oscillation of the three different neutrinos flavors – electron-, muon- and tau-neutrino – into each other, a phenomenon called neutrino oscillation. The frequency of this oscillation is determined by the mass squared differences $\Delta m_{ij}^2 = m_i^2 - m_j^2$. The mass ordering of two mass eigenstates $m_2 > m_1$ has already been determined, leaving an open question regarding the ordering of the third mass eigenstate to be answered.

The Jiangmen Underground Neutrino Observatory (JUNO) is a neutrino experiment currently under construction in Jiangmen, China, scheduled to begin taking data in 2020. Its main goal is to measure the sign of the mass difference $\Delta m_{31}^2$ of the neutrino mass hierarchy. [1]

The main channel for the detection of neutrinos in this experiment is the interaction of electron antineutrinos ($\bar{\nu}_e$) via Inverse Beta Decay (IBD) $\bar{\nu}_e + p \rightarrow n + e^{+}$ with protons in the detector medium, resulting in a positron and a neutron. As positrons ($e^{+}$) and electrons ($e^{-}$) signatures produce similar signatures in a liquid scintillator detector such as JUNO, there is a significant amount of background, especially from the decay of $^9$Li and $^8$He atoms, whose ($\beta^{-} + n$)-decay channel mimics IBD events [2]. A successful classification of signatures in the detector of those particles would lead to a reduction of background for this detection channel. [3]

Convolutional neural networks (CNNs) have been shown to achieve remarkable successes in the classification of images and other highly non-linear tasks and have already been used successfully for the identification of particle interactions in detectors built for neutrino physics [4].

In this thesis the use of CNNs for $e^{+}/e^{-}$ discrimination by determining shape and timing differences in the data is investigated, utilising a self developed method for the generation of 2D input images from spherical detector data based on the HEALPix algorithm traditionally used in cosmology.

It starts by giving a short introduction into the theory of neutrino oscillation, the design and purpose of the JUNO detector and the inverse beta decay and its background in JUNO. Then, a short overview over the basic principle of neural networks and convolutional neural networks is given, and the properties of the simulation data used in this thesis are discussed. A method of converting detector data in to two dimensional images using HEALPix is introduced, and a CNN is trained on the image dataset and the results and their dependence on initial particle position in the detector, deposited charge and visual energy are examined.
2 Theory of neutrino oscillation

As of today there are three known flavours of neutrinos: The electron-, muon and tau-neutrino. These lepton flavour states can be expressed as a superposition of three neutrino eigenstates of definite mass. As neutrinos propagate through space, the phase of these different eigenstates changes at a differing rate, leading to a change in neutrino flavour. This phenomenon is called neutrino oscillation.

For the phases of the mass eigenstates to advance at different rates, they need to have slightly different masses. The sign of the squared mass differences $\Delta m^2_{ij}$ determines the hierarchy of the masses, with the sign for $\Delta m^2_{21}$ already determined to be positive. This results in an ordering for the lower two masses of $m_2 > m_1$.

The sign of $\Delta m^2_{31} (\approx \Delta m^2_{32})$ has yet to be determined. The scenarios for the two different possible signs are called normal mass hierarchy (NH) and inverted mass hierarchy (IH).

The determination of the type of hierarchy is the main goal of the JUNO detector. Figure 1 shows the small difference in the reactor antineutrino IBD prompt energy spectrum for the different mass hierarchies. To resolve this difference and use it to determine the neutrino mass hierarchy, the JUNO collaboration aims to have an energy resolution of at least $3\%$ MeV$^{-1}$. [1]

Figure 1: Estimation of positron energy spectrum of the inverse beta decay reaction for a JUNO-like idealized reactor antineutrino experiment (20 kt, 40 GW nuclear reactor at 58 km, 5 years operation. The different mass hierarchies, NH in blue and IH in red, result in a phase shift of the high-frequency oscillation. The hierarchy was inverted by changing the sign of $\Delta m^2_{31}$. [5]
3 Jiangmen Underground Neutrino Observatory (JUNO)

The Jiangmen Underground Neutrino Observatory is a multi-purpose liquid scintillator detector of 20 kt fiducial volume currently under construction in Jiangmen, China, whose main scientific goal is the determination of the neutrino mass hierarchy. Built at a distance of 53 km from two nuclear power plants with a combined power of 36 GW, it aims to determine the mass hierarchy at a significance of 3-4 $\sigma$ after six years of operation using the reactor neutrino flux. The main detection channel of those reactor-antineutrinos is the Inverse Beta Decay (IBD), where an electron antineutrino interacts with a proton to produce a positron and a neutron. \[1\]

3.1 Detector setup

One of the more current detector designs is shown in Figure 2. \[2\]

The central detector consists of around 20 kt of liquid scintillator on the basis of linear alkylbenzene (LAB) contained in a spherical container with a radius of 17.7 m, drawn in yellow in Figure 2. Some particle interaction excite the LS and cause the production of scintillation light, which can be detected by about 18,000 20 inch and 25,000 3 inch photomultiplier tubes (PMTs) installed more or less uniformly at a radius of 19.5 m around the sphere. The scintillator’s light yield is 1200 photoelectrons (PE) per MeV deposited energy. To shield the LS from radioactive decays in the glass cover of the PMTs, the PMTs are seperated from the LS by a waterbuffer, drawn in blue in Figure 2. \[2\]

This whole inner detector is submerged in a cylindrical pool of ultra-clear water equipped with additional PMTs, forming the water Cerenkov detector. This pool serves a dual purpose: It shields the inner detector from the surrounding environmental radioactivity and enables the detection of charged particles, such as atmospheric muons, which
are responsible for the most seriously correlated background to reactor antineutrinos by
producing cosmogenic isotopes such as $^9\text{Li}$ and $^8\text{He}$.\textsuperscript{1} Additionally, a muon tracker formed of plastic scintillator strips will be installed on top of the water pool to enable further vetoing of muons by providing high precision muon track starting points.\textsuperscript{1,2}

### 3.2 Inverse Beta Decay detection channel

The main detection channel for reactor neutrinos in JUNO is the Inverse Beta Decay (IBD) shown in figure 3. Here, the incoming reactor antineutrino $\bar{\nu}_e$ scatters off a proton in the scintillator via charged current interaction, creating a positron and a neutron.\textsuperscript{8}

$$\bar{\nu}_e + p \rightarrow n + e^+$$

Due to the mass difference between positron and neutron, the positron carries almost all the energy of the incoming antineutrino after scattering\textsuperscript{8,9}. The positron deposits its energy by scattering through the scintillator medium, creating a cascade of scintillation light in the process, and finally annihilates, after losing all its kinetic energy, with an electron into two 511 keV $\gamma$-rays emitted back-to-back. The neutron thermalizes in the detector, after which it is captured by a proton around 200 $\mu$s later, emitting a 2.2 MeV $\gamma$-ray. Figure 4 shows a schematic of the IBD signature in the LS.\textsuperscript{1} Due to the instantaneous creation of light caused by the annihilation of the positron, this signal is often referred to as the $prompt$–signal, while the scintillation light emitted by

\[\begin{array}{c}
\text{p} \\
\downarrow \\
W \\
\downarrow \\
\bar{\nu}_e \\
\downarrow \\
e^+ \\
\end{array}\]

Figure 3: Leading order feynman diagram of the Inverse Beta Decay
the neutron capture is called the delayed-signal. The delayed coincidence of this signal pair leads to an identifiable signature, causing it to be a commonly used process for the identification of electron antineutrinos.

3.3 Background of the Inverse Beta Decay

With reactors of 36 GW thermal power at a baseline of 53 km, the expected rate of IBD events for the LS detector with a fiducial volume of 20 kt is 83 per day [1]. There are 5 different classes of background noise to be taken into consideration when looking for IBD events (as described by Fengpeng, A. et al.) [1]:

Accidental background. Accidental background events are random coincidences of unrelated events that have a similar signature to IBD events. This background consists mainly of three different types:

- Radioactive decay + Radioactive decay: Natural radioactive decays caused by radioactive isotopes in the surrounding rock, in the detector parts or impurities in the LS. The singles rate, a single being one signal in the detector that resembles part of the IBD event, obtained from a MC-Simulation is about 7.6 Hz, with neutron-like signals accounting for 8%. The expected rate of prompt-delayed coincidence signals within 1.0 ms is 410/day.
• Radioactive decay + Cosmogenic Isotopes: Radioactive isotopes generated by atmospheric muon showers cause about 340 neutron-like single events per day. The resulting rate for prompt-delayed coincidence signals within 1.0 ms is <0.01/day.

• Radioactive decay + Spallation Neutrons: Spallation neutrons are generated by atmospheric muons inelastically scattering off the surrounding rock and detector. The total expected rate of spallation neutrons is 1.8 Hz. After time and spatial cut, the resulting rate of prompt-delayed coincidence signals between natural occurring radioactive decays and the Spallation Neutrons is negligible.

**Geo-neutrino background.** Electron antineutrinos caused by radioactive decays of thorium and uranium inside the earth constitute the geo-neutrino flux, which contributes to the reactor-antineutrino background. The total event rate for Geo-neutrinos is estimated from MC-Simulations to be 1.5/day.

**Fast neutron background.** Muons unable to be tagged by the veto detectors, be it by only passing the surrounding rock of the water pool or only clipping the corner of the water pool, produce energetic neutrons, which can form coincidence-like signals by scattering elastically in the detector and creating scintillation light, until being captured by a proton. The estimated rate of these fast neutron events is 0.1/day.

**(α-n) background.** Another source of background signals are alpha particles created by the decay of uranium or thorium atoms reacting with the $^{13}\text{C}$ in the LS.

\[ \alpha + ^{13}\text{C} \rightarrow n + ^{16}\text{O}^{(*)} \]

If the resulting neutron is fast enough, or the de-excitation of the $^{16}\text{O}^{*}$ emits a $\gamma$-ray, this can lead to a correlated background. The expected rate for this kind of background events is 0.05/day.

**$^9\text{Li}/^8\text{He}$.** Cosmogenic $^8\text{He}$ and $^9\text{Li}$ constitute the most seriously obstructive type of background event, due to the possibility of ($\beta^-$-n) decay (branching ratio of 16 % and 51 % respectively).

\[ ^9\text{Li} \rightarrow \beta^- + n + ^8\text{Be} \]

In the detector this decay channel mimics IBD events, since the resulting $^8\text{Be}/^7\text{Li}$ atom carries almost no kinetic energy, and thus does not show in the detector, and electron signatures look very similar to positron signatures, with the only difference being the additional electron annihilation of the positron. The expected rate of ($\beta^-$-n) decays is 84/day, with the prediction of the rate based on the cross sections measured in the KamLAND detector [1].
3.4 Positron-electron discrimination

As mentioned in the previous chapter, electrons and positrons create similar signatures in the detector. The only difference is the additional annihilation signal for the positron. Besides annihilating directly with an electron, there is also the possibility that the positron forms a short-lived positronium together with an electron. This positronium has two possible ground states: A singlet state with antiparallel spins, called para-positronium, and a triplet state with parallel spins called ortho-positronium. Ortho-positronium was not included in the simulation data for this thesis. The para-positronium’s lifetime in vacuum is 0.125 ns and it can decay into any even number of photons, but the branching ratio for decays with a number of photons larger than 2 is negligible. \[10,11\] Ortho-positronium is more long-lived, having a lifetime of approximately 142 ns in vacuum. Its main decay channel is a decay into 3 photons, but it is able to decay into any uneven number of photons. Branching ratio for any decay for a number of photons greater than 3 is negligible. \[11\]

Due to interactions with the scintillator medium, the lifetime of positronium is significantly reduced. This does not matter for the para-positronium, since its lifetime is too small compared to the spatial resolution of the Detector. The lifetime of the ortho-positronium is shortened to about 3 ns. It can also lose angular momentum by interacting with the medium, which leads to the ortho-positronium mainly decaying into two photons instead of three \[12\]. The resulting photons of the annihilation, be it after forming a positronium or direct annihilation, typically travel about 24 cm in the detector, after which they interact with the LS, creating an electromagnetic cascade resulting in additional scintillation light \[13\].

In summary, there are three possibly observable differences between positrons and electrons in the JUNO detector:

- Difference in the distribution of light across the detector sphere
- Difference in PMT hit times
- Larger difference in PMT hit times for ortho-positronium

These differences are so small compared to the time and spatial resolution of current detectors that classic analysis methods struggle with their reliable detection. In this thesis a relatively new approach is examined by trying to train neural networks on these minute differences by using the observed data from the PMTs. Only the difference due to ortho-positroniums is exempt from training, since the simulation used to create the data set for this thesis is as of yet unable to simulate ortho-positroniums.
4 Deep Learning

4.1 Basic principles of neural networks

Figure 5: Examplary MLP structure with 2 hidden layers with 3 nodes each

Deep learning describes a subset of machine learning methods based on learning data representations in a way that is vaguely inspired by the way biological nervous systems process information. One of those methods is the use of neural networks. The most common and or basic of such neural networks is the MultiLayer-Perceptron (MLP), also often called Dense Neural Network (DNN) \[14\]. The structure of an MLP is as follows: A one-dimensional input layer that represents the input we are trying to classify, followed by one or more hidden layers that transform the input from the layer before according to specific variables, and a final output layer, where each node maps to a different output metric devised by the creator of the network. An examplary simple structure of a neural network is shown in Figure 5. The ability of MLPs to transform the input stems from nodes that are often called perceptrons – since they are roughly modeled after biological neurons – and follow a simple principle:

A perceptron receives multiple inputs \( I_n \), weighs them individually with weights \( w_n \) and adds the weighted inputs up. After that a constant offset called bias \( b \) is added and the total is passed to an activation function \( \sigma \). The resulting output \( y \) is then passed on. Summarized, this can be written as:

\[
y = \sigma \left( \sum w_i x_i + b \right)
\]
Figure 6 shows the schematic of such a perceptron.

![Neuron schematic](image)

Figure 6: A single node in an MLP, often called perceptron.

In an MLP, every layer \( L_i \) has a number of nodes \( N_i \) and every node of a layer is connected to all nodes of the previous layer. It can be shown that MLPs are able to approximate any function to arbitrary precision given a sufficient number of hidden nodes \[15\].

The weights and biases used for the solution of the problem are typically determined using supervised learning. During supervised learning, the MLP is presented input examples for which the corresponding true output is known. Then the loss, a metric for the deviation of the output of the network from the known value, is calculated. Often used loss functions include mean square error, binary cross-entropy and categorical cross-entropy. Using the back-propagation algorithm, as described in \[16\], the gradient of the network’s weights and biases is calculated. Then, the loss is minimized by altering weights and biases using stochastic gradient descent or other similar gradient-based optimizers and repeated until the deviation of the network output from the true output is reduced to an acceptable level. \[16–18\]

Even though the MLP is a powerful tool, it has numerous deficiencies when working with pictures. Firstly, MLPs completely ignore the topology of the input, as the input carries no information about the position of the various pixels and their distance and direction from each other. Secondly, even when an MLP learns about this spatial correlation, it is still not translationally/rotationally invariant, so it has to learn the same thing for each translation/rotation of a feature. An MLP can potentially learn such features using brute force and a large number of nodes/layers, but then it still has not learned any of the image features it needs for the classification of the images. Hence, MLPs often need to have a large number of layers and trainable parameters, leading to unnecessarily long training times. Besides, this may cause overtraining. Overtraining, also called overfitting, is a problem that occurs during the training of a neural network, where the network starts to train on sampling noise existing only in the dataset the network is trained on, but not in real test data. This mostly occurs when a network is trained for too long or the complexity of the network and its amount of free trainable parameters
is large compared to the dataset it is trained on. 

4.2 Convolutional neural networks

Convolutional neural networks (CNN) include convolutional filters instead of simple perceptrons, such as in Figure 6. These filters are also often called convolutional kernels and are also grouped in layers, with consecutive feedforward layers constituting a CNN. Convolutional kernel work the following way: For each convolutional kernel in a layer, an inner product between the filter and all regions in the picture is calculated and fed to an activation function, see Figure 7, producing a feature map of the input picture. The feature map of a convolutional kernel encodes the similarity of all sections of an input image filter with the feature corresponding to the filter. The different feature maps of the several filters are stacked on top of each other, forming a new multi-dimensional picture which is, in a way, a combination of many alternative representations of the input image. This picture is then passed on to the next convolutional layer. 

To reduce the dimensionality of images in a network, often necessary to reduce computational time, so-called Pooling Layers are applied. In this thesis two different pooling techniques were used: Max Pooling, where the values in the receptive field of a pooling kernel are reduced to their maximum and Average Pooling, where the values in the receptive field of the pooling kernel are reduced to their average, see Figure 8. CNNs remedy the problems mentioned in chapter 4.1 in some way. Since their architecture assumes spatial correlation between neighboring pixels and conserves this information when passing feature maps on, the network does not need to learn spatial correlation by itself. This means that CNNs are translationally invariant due to their architecture. This leads to CNNs outperforming MLPs in tasks of image classification. 

The weights of the filters will be updated during training in the same way as in an MLP, allowing the same training process for CNNs, and also the combination of both types of networks.
Figure 8: 2x2 Pooling example on a random feature map [7].
5 Simulation data

![Distribution in visible charge for the whole dataset.](image)

Figure 9: Distribution in visible charge for the whole dataset. The distributions for positrons (red) and electrons (blue) are slightly different, with a slightly lower average visual charge value for positrons. Noteworthy is the low charge value tail for positrons.

In order to obtain enough data for the training of a neural network, simulation data generated by a Monte Carlo (MC) simulation of the detector provided by the JUNO framework\footnote{The Software version is J17v1r1} is used in this thesis. This simulational data includes only high-level data, such as hit times of PMTs, their seen charge and initial generation information, stored in a ROOT dataformat (more about ROOT and the ROOT dataformat in \cite{21}).

The initial generation information includes the particle type, electron or positron, the initial position of this particle in cartesian coordinates and the particle momentum in MeV.

The generated dataset contains 100,000 electrons and positrons each, equally distributed in initial position in the detector, compare Figure \ref{fig:10} and following nearly the same distribution in total visual charge – total visual charge denotes the total amount of detected photons per event. Unit of the visual charge in this thesis is photoelectron PE, with one PE being equivalent to the charge registered by a PMT when hit by one photon. To achieve this distribution in total visual charge, the electrons need to be shifted to a lower energy range, to compensate for simulational effects that lead to
Figure 10: Distribution of initial positions for the whole dataset

Figure 11: Distribution in visual energy for the whole dataset
positrons depositing less charge per MeV visual energy than electrons. For electrons, the visual energy is equally distributed between 0.92 MeV and 1.32 MeV, with positron visual energies shifted by around 0.10 MeV upwards. 

Visual energy is the energy that is deposited in a detector by a particle in processes that lead to the creation of scintillation light, hence 'visible'. The visual energy of positrons is 1.022 MeV higher than for electrons with the same momentum, due to the scintillation light that is emitted by the annihilation of the positron with an electron.

The same distribution in charge for positrons and electrons is necessary to prevent the network from learning to differentiate by looking at the total charge value of the event, instead of learning differences in shape.

In this thesis, only the information of the 20 inch PMTs was used, and a time cut of < 300 ns was applied, as late hits tend to undergo multiple scatterings. Two values were used for the later generation of images in this thesis:

- **charge value** per PMT, which is a measurement for the amount of photons that hit this PMT.

- **hit times** per PMT, which are a measurement for the timing of the photon hits on this PMT.
6 Image production

Since most existing network architectures and frameworks for CNNs are designed for 2D-image classification tasks and not for spherical images, to fully make use of the existing structures, a conversion from spherical data to 2D images is needed.

6.1 Previous conversion process

A solution for the conversion applied by Birkenfeld [7] is the projection of the PMTs on the sphere into the $(\theta, \phi)$-space. Further steps are: Rotation of the event into the center of gravity (CoG) of the charge to avoid splitting the image at the edges and reducing the distance between pixels in $\theta$-direction to get rid of pixels without a corresponding PMT. The whole process is shown in Figure 30 in Appendix A. [7]

The resulting images carry information per PMT per pixel. Each pixel contains 1 or 3 channels, depending on configuration, in which information is stored. Birkenfeld [7] used combinations of charge value, first hit time, mean hit time and standard deviation of hit times. Training on these kind of pictures has been inconclusive [7]. This could be due to the way the data is converted into images, since every conversion distorts spatial properties, such as area, shape or direction, at various levels of severity. These distortions might obstruct or slow training for specific conversions.

In this thesis a different approach to image production was taken by using the Hierarchical Equal-Area and Isolatitude Pixelization (HEALPix) algorithm to divide the surface of the detector-sphere into areas of equal size, see Figure 12. The data of the PMTs in each of those pixels is then mapped onto a 2D image modeled after the base HEALPix grid.

6.2 Healpix Algorithm

HEALPix, introduced by Gorski et al. [22], offers a way of distributing $12N^2$ ($N \in \mathbb{N}$) points as uniformly as possible over the surface of a sphere under the constraint that the points lie on a small number of parallels of latitude ($4N-1$) and are equidistant in azimuth (on each ring). HEALPix also specifies non-geodic boundaries between adjacent points in a way that results in an equal-area pixelization of the sphere. Figure 12 shows the distributed points and their corresponding boundaries for different $N$.

The base resolution ($N=1$) of this HEALPix pixelization comprises twelve pixels in three rings around equator and poles. For higher resolutions ($N>1$) these 12 base resolution pixels are each subdivided into $N^2$ quadrilaterals of equal area. [18]

The HEALPix data is stored as a 1D array. The allocation of these values to the HEALPix grid follows one of the following schemes:

RING scheme, in which the pixels are simply numbered in ascending order along each iso-latitude ring moving from north to south pole. This scheme supports all $N \in \mathbb{N}$ and is best suited to spherical harmonic analysis.

NESTED scheme, in which the pixel indices are arranged in twelve tree structures corresponding to the base-resolution pixels, see Figure 13. This scheme only supports
Figure 12: HEALPix partition of the sphere for $N=1,2,4,8$ (in clockwise order) [22].

Figure 13: HEALPix nested tree scheme. On the left, one HEALPix base pixel for $N=2$ is shown. When increasing resolution, every pixel splits into 4 daughter pixels, which inherit the pixel index of their parent and acquire two new bits to give a new pixel index. [22]
Figure 14: Healpix ordering schemes. The top picture shows the RING ordering scheme, the bottom picture shows NESTED ordering scheme. [18]

\[ N \in 2^N \] and is best for all analyses involving nearest-neighbor searches. Figure [14] shows both ordering schemes for N=2. For this thesis the RING scheme was used because of the larger amount of usable resolutions.

6.3 Image mapping using HEALPix

In this thesis a Python-module called HEALPy\(^2\) was used, which implements HEALPix and many useful functions for the generation and analysis of HEALPix maps, such as assignment of angular coordinates to HEALPix-pixels, different pixel queries and other analysis tools.

For the conversion of the raw event data from the ROOT file, which stores all hits per PMT and the PMT coordinates, to a 2D image, the following steps were taken (in order):

- Calculating the CoG for each event and rotating the coordinate system, such that the CoG lies at angular coordinates \((\pi/2,0)\).

- Using HEALPy to assign PMTs to their corresponding HEALPix-pixels.

- Creating two HEALPix arrays, one with the average charge of all PMTs corresponding to a pixel, and one with the average first hit time of the PMTs in a pixel.

\(^2\)https://healpy.readthedocs.io/en/latest/
Figure 15: Exemplary image of charge distribution over the detector sphere for a positron event, visualized in a Mollweide projection. The same event is shown in 4 different resolutions, corresponding to a HEALPix resolution parameter \( N \) of 1,4,8,12 (clockwise).

Figure 16: The amount of PMTs per pixel for a resolution parameter of \( N=12 \) taken over 100 events.
Figure 17: Conversion for different HEALPix resolution parameters $N$. $N=1$ shows the conversion of the base pixels. All white pixels are filled with zeros in the conversion process, to achieve a rectangular image.

- Normalizing the charge array by the total charge of the event to prevent the network from learning to differentiate between positron and electrons by total visual charge – necessary because the distributions for electrons and positrons are not exactly the same, see figure 9.

Figure 15 shows the resulting images for the charge channel, using the built-in HEALPy function to regrid the data to a Mollweide Projection. The average amount of PMTs per pixel for a resolution parameter of $N=12$ is 10.2, Figure 16 shows the distribution of PMTs per pixel at $N=12$ averaged over 100 events.

### 6.4 Conversion of HEALPix maps to 2D images

Figure 17 shows a way to map HEALPix-maps to a 2D grid, without regridding and the potential introduction of artefacts, inspired by a projection by Calabretta and Roukema [18]. Here, the pole-side edges of the base pixels at the poles are disconnected from their
neighboring pixels. Then, the resulting grid comprised of twelve base pixels is rotated by 45° and then converted into a 2D rectangular image by padding all necessary pixels with zeros. The resulting image plane is exactly half filled, with the center of the picture taken by a 2x2 base grid, in which the event center is situated. To achieve this, the HEALPix map of the events needs to be rotated by $\pi/4$ in longitudinal direction before conversion. This kind of conversion introduces loss of spatial information around the edges, but this should only impact network performance slightly, since most events are concentrated in the center 2x2 base grid.

6.5 Visualizing shape differences between $e^+$ and $e^-$

To search for visual shape differences between positrons and electrons that a CNN might be trained on, the CoG-centered images for all positron and electron events respectively were added up and the ratios for the visual charge channel computed, see Figure 18 for the outcome.
A small abundance of electron charge in the center of the event is visible, which could be a feature that a neural network might train on.
7 Application of Deep Learning to $e^+/e^-$ discrimination

For this thesis, the Keras API \cite{23} for Python, which implements many of the necessary algorithms for the construction, training and evaluation of neural networks, was used. As a backend for Keras, Google Brains Tensorflow \cite{24} was used, enabling the fast training of MLPs and CNNs on Graphical Processing Units (GPUs).

7.1 Network Architecture

Figure 19: The variation of the inception module used in this thesis. The convolutional filters of differing sizes enable the network to apply different receptive fields in one layer.

The CNN architecture constructed in Keras was inspired by the prize-winning GoogLeNet architecture \cite{25}. Centerpiece of this network architecture is the inception module, a network-in-network approach aimed at improving the utilization of computing resources and applying different receptive fields in one layer (combining 1x1, 3x3 and 5x5 convolutional kernel) \cite{25}. The structure of the abridged inception module used in this thesis is shown in Figure 19. The total number of parameters and filters per layer can be taken from Figure 31 in Appendix A.

Since the input images used in this thesis are less complex than most natural images, that the full GoogLeNet architecture trained on, and there are only two different classes in our network, it was found that any amount of inception layers above five lead to no accuracy improvement.

The activation function used for all convolutional layers is the rectified linear unit (ReLU)

$$y_i = \max(0, x_i),$$
as at present, ReLU is the most popular non-linear function for use in CNNs and allows for training of deep supervised networks without unsupervised pre-training. The output of the final inception layer was down-sampled using average pooling, before being passed to the output layer consisting of two classifier nodes using the softmax function to achieve an added-up sum for the classifier outputs of one. The softmax function is defined as followed:

$$\sigma_i(y) = \frac{\exp(y_i)}{\sum_i \exp(y_i)}$$

where $y$ is the activation of the final output nodes before the softmax function $\sigma_i$ is applied.

The complete network architecture used for this thesis is shown in Figure 20. Gaussian Noise, Batch Normalization and Dropout are regularization techniques and further explained in chapter 7.2.

![Network Architecture](image)

Figure 20: Network architecture designed for this thesis

### 7.2 Training of the network

**Training process** For training, the dataset was split into three separate parts, with 72% of events used for training, 8% for validation of the training behavior and 20% used for testing the trained network. Making use of mini-batch training, as described in [27], to simultaneously train on 32 training examples each iteration, the final softmax output was used to calculate categorical cross-entropy loss by comparing the output for each batch to its target value.

To avoid overfitting and to ensure that the trained model generalizes beyond the training sample, multiple regularization techniques were used: Penalty terms proportional to the square of the weights in each kernel were included in all convolutional layers, preventing the weights from becoming too large and reacting too strongly to a single input feature (a technique called L2-regularization). This leads to an additional term in the computed loss:

$$loss = \text{categorical crossentropy} + \lambda \sum w_i^2$$
with the regularization parameter $\lambda$ and all regularized weights $w_i$. Additionally, the Dropout technique was employed on several layers in the network, in which each weight of a layer is set to zero with a certain probability at each iteration. This ensures that every trainings iteration uses a random subsample of the network, leading to a final network that is essentially an ensemble of multiple smaller networks, which usually performs better.\[20\]

Furthermore, to augment the amount of data the network can train on, Gaussian noise with a standard deviation of 1% was added to vary pixel intensities to hinder the network from relying too heavily on the intensity of single pixels. After the noise layer applying noise to each input batch, a batch normalization layer was added. Batch normalization is a technique used to perform normalization for each training mini-batch, allowing for higher learning rate and acts as a regularize\[28\].

Suitable values for a few hyperparameters, such as L2 regularization weight, Dropout ratio, Batch size and Gaussian Noise level were determined using a grid search, where different hyperparameter configurations were qualitatively examined over 20 epochs of training.

Stochastic gradient descent was used as an optimizer for training, with the learning rate being reduced every epoch by 1% to achieve a better convergence. At first, the network was trained on datasets with differing HEALPix resolutions from N=3 to N=12. Here, due to time constraints, the network was only trained over 40 and 60 epochs with reduced L2-penalty term and a reduced dropout percentage of 20% to ensure faster convergence. The accuracy of the network on the test set was evaluated for all resolutions and a suitable resolution of N=12 was determined for further, more exact training with higher regularization terms.

This training was carried out over 150 epochs. Figure 21 shows loss and accuracy for training and validation samples as a function of trained epochs, where each epoch consists of 9,000 batch iterations. The similar loss on both samples up to where the accuracy for training and validation dataset diverge suggests that the network found working generalizations up to about 57-58%.

### 7.3 Results

Figure 22 shows the network accuracy for different resolutions. The achieved network accuracy increases with higher resolutions. Due to memory constraints of the GPU, no resolutions corresponding to HEALPix resolution parameter N > 12 could be tested. The accuracy is calculated by dividing the number of correctly identified events by the total amount of events in the test dataset.

$$\text{accuracy} = \frac{\#\text{correctly identified events}}{\#\text{all events}}$$

Further, more heavily regularized and longer training of the network on the whole dataset for a HEALPix resolution parameter of N=12 reached a total test accuracy of 58.0%. Figures 23, 24 and 25 show the accuracy as a function of total charge value, initial particle position and visual energy for this network.
Figure 21: Loss calculated on the training dataset (blue) and the validation dataset (green).

Figure 22: Network accuracy as a function of the used HEALPix resolution parameter N. The network described in chapter 7.1 was trained with reduced regularization terms over 40 (blue) and 60 (green) epochs and evaluated on the test dataset for each resolution.
Figure 23: Network accuracy as a function of total charge on the test data set. The red dashed line indicates the baseline for a random classifier. Share of positrons denotes the percentage of positrons in the corresponding bin.

Figure 24: Network accuracy as a function of initial particle position in the detector on the test data set. The red dashed line indicates the baseline for a random classifier. Share of positrons denotes the percentage of positrons in the corresponding bin.
Figure 25: Network accuracy as a function of visual energy on the test data set. The red dashed line indicates the baseline for a random classifier. Share of positrons denotes the percentage of positrons in the corresponding bin.

Figure 26 shows that the network was able to learn the peculiarities of the PE distribution, tagging all low-charge events as positrons and the majority of high-charge events as electrons, thus making use of the distribution’s edge effects.

Figure 24 shows that the accuracy is steadily increasing towards the border of the detector until it peaks at roughly 73.3% for events at a distance from the center of the detector over 17.1 m.

Figure 27 shows the network accuracy as a function of initial particle position and total charge value. Noticeably, the positron events with especially low total charge value all occur at a radius $R > 16.76$ m. This could be due to them being events where one photon generated by the annihilation of positron and electron escapes the LS without creating scintillation light, resulting in a lower total charge value. Because these are the same events the network trains on when making use of the distributions edge effects, this could partially explain the increased accuracy for events further from the center of the detector. When applying a fiducial volume cut with $R < 16.76$ m to exclude all events with a charge value lower than 1100 PE, the network accuracy decreases to 57.3%.

Since the output of the final Softmax layer of the network is normalized, the scores output of the layer can be roughly interpreted as a probability calculated by the network of the input event falling into one of the two particle classes. Figure 28 shows the distribution of the $e^+$-classifier score for all test events. The fact that there is a huge overlap in positron classifier scores for both electrons and positrons shows how difficult positron-electron discrimination is for the network.

Figure 29 shows the cumulative efficiency and purity depending on the positron classifier
Figure 26: The top graph shows the purity depending on the total charge PE, the bottom graph the percentage of events for a charge value that the network predicted to be positrons (red) or electrons (blue). The network noticeably more often predicts positrons for a lower total PE than for a higher, showing that it is able to train on the edge effects of the PE distribution.

Score. Cumulative efficiency of a network is the amount of correctly identified relative to the total amount of positrons for all events above a particular positron classifier score, while cumulative purity is the ratio of correctly identified positrons to all events identified as positrons above the particular positron classifier score. The product of both is then the total percentage of correctly identified positrons when tagging all events above a particular positron classifier score.
Figure 27: Network accuracy as a function of total visual charge and initial particle position on the test data set.

Figure 28: Distribution in $e^+$ score for positrons (red) and electrons (blue)
Figure 29: Cumulative efficiency (red), purity (blue) and their product (green) when selecting all events above a particular classification score. Efficiency denotes the amount of correctly identified positrons relative to the total amount of positrons, Purity the amount of correctly identified positrons relative to the amount of all events identified as positrons by the network.
8 Conclusion and outlook

In this thesis an approach to electron-positron discrimination using convolutional neural networks was presented. For this purpose, an algorithm for the conversion of spherical detector data to 2D images using HEALPix was introduced. The analysis was carried out on 200,000 particle events produced by a Monte Carlo simulation of the interaction of positron and electrons with the JUNO detector.

Comparing the average images for electron and positron, a small difference in charge distribution was found, with the charge in electron images being more concentrated on average.

A neural network of seven convolutional layers, following a similar network in network approach as GoogLeNet [25], was evaluated on the data and the influence of input image resolution, initial particle position, visual energy and total charge value on the accuracy of the trained network was examined.

Results have shown to be yet inconclusive, as the best achievable total accuracy of the network has been a low $58\%$. A dependency on initial particle position in the detector was observed, with events further than 16.76 m from the center of the detector being correctly identified with an accuracy of $73.3\%$. It was also observed that the accuracy of the network relies on edge effects in the available simualtional data, which are especially prevalent for events far from the center of the detector. A fiducial volume cut $R < 16.76$ m eliminating a part of those edge cases lead to a decreased accuracy of $57.3\%$.

Even though the low accuracy seems to make a sucessful approach to electron-positron discrimination using neural networks to train on the distribution of light on the detector sphere implausible, training on augmented or different datasets might lead to different results. Training on simualtional data that includes ortho-positronium in its calculations or training on datasets including data from an electronics simulation might include enough features separating electrons from positrons that a neural-network-based approach might prove to be more sucessful.
Figure 30: Previous conversion solution
<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
<th>Connected to</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_1 (InputLayer)</td>
<td>(None, 54, 36, 2)</td>
<td>0</td>
<td>input_1[0][0]</td>
</tr>
<tr>
<td>batch_normalization_1</td>
<td>(None, 54, 36, 2)</td>
<td>8</td>
<td>input_1[0][0]</td>
</tr>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 54, 36, 128)</td>
<td>12672</td>
<td>batch_normalization_1[0][0]</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 54, 36, 128)</td>
<td>0</td>
<td>conv2d_1[0][0]</td>
</tr>
<tr>
<td>max_pooling2d_1 (MaxPooling2D)</td>
<td>(None, 26, 17, 128)</td>
<td>0</td>
<td>dropout_1[0][0]</td>
</tr>
<tr>
<td>conv2d_2 (Conv2D)</td>
<td>(None, 26, 17, 16)</td>
<td>2864</td>
<td>max_pooling2d_1[0][0]</td>
</tr>
<tr>
<td>conv2d_3 (Conv2D)</td>
<td>(None, 26, 17, 128)</td>
<td>18560</td>
<td>conv2d_2[0][0]</td>
</tr>
<tr>
<td>dropout_2 (Dropout)</td>
<td>(None, 26, 17, 128)</td>
<td>0</td>
<td>conv2d_3[0][0]</td>
</tr>
<tr>
<td>max_pooling2d_2 (MaxPooling2D)</td>
<td>(None, 12, 8, 128)</td>
<td>0</td>
<td>dropout_2[0][0]</td>
</tr>
<tr>
<td>ltn1_1 (Conv2D)</td>
<td>(None, 12, 8, 32)</td>
<td>4128</td>
<td>max_pooling2d_2[0][0]</td>
</tr>
<tr>
<td>ltn1_2 (Conv2D)</td>
<td>(None, 12, 8, 32)</td>
<td>4128</td>
<td>max_pooling2d_2[0][0]</td>
</tr>
<tr>
<td>ltn1_3 (MaxPooling2D)</td>
<td>(None, 12, 8, 128)</td>
<td>0</td>
<td>max_pooling2d_2[0][0]</td>
</tr>
<tr>
<td>otn1_1 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>18496</td>
<td>ltn1_1[0][0]</td>
</tr>
<tr>
<td>otn1_2 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>51264</td>
<td>ltn1_2[0][0]</td>
</tr>
<tr>
<td>otn1_3 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>8256</td>
<td>ltn1_3[0][0]</td>
</tr>
<tr>
<td>concatenate_1 (Concatenate)</td>
<td>(None, 12, 8, 192)</td>
<td>0</td>
<td>otn1_1[0][0]</td>
</tr>
<tr>
<td>ltn2_1 (Conv2D)</td>
<td>(None, 12, 8, 32)</td>
<td>6176</td>
<td>concatenate_1[0][0]</td>
</tr>
<tr>
<td>ltn2_2 (Conv2D)</td>
<td>(None, 12, 8, 32)</td>
<td>6176</td>
<td>concatenate_1[0][0]</td>
</tr>
<tr>
<td>ltn2_3 (MaxPooling2D)</td>
<td>(None, 12, 8, 192)</td>
<td>0</td>
<td>concatenate_1[0][0]</td>
</tr>
<tr>
<td>otn2_1 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>18496</td>
<td>ltn2_1[0][0]</td>
</tr>
<tr>
<td>otn2_2 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>51264</td>
<td>ltn2_2[0][0]</td>
</tr>
<tr>
<td>otn2_3 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>12352</td>
<td>ltn2_3[0][0]</td>
</tr>
<tr>
<td>concatenate_3 (Concatenate)</td>
<td>(None, 12, 8, 192)</td>
<td>0</td>
<td>otn2_1[0][0]</td>
</tr>
<tr>
<td>ltn3_1 (Conv2D)</td>
<td>(None, 12, 8, 32)</td>
<td>6176</td>
<td>concatenate_3[0][0]</td>
</tr>
<tr>
<td>ltn3_2 (MaxPooling2D)</td>
<td>(None, 12, 8, 192)</td>
<td>0</td>
<td>concatenate_3[0][0]</td>
</tr>
<tr>
<td>otn3_1 (Conv2D)</td>
<td>(None, 12, 8, 128)</td>
<td>36992</td>
<td>ltn3_1[0][0]</td>
</tr>
<tr>
<td>otn3_2 (Conv2D)</td>
<td>(None, 12, 8, 64)</td>
<td>12352</td>
<td>ltn3_2[0][0]</td>
</tr>
<tr>
<td>concatenate_4 (Concatenate)</td>
<td>(None, 12, 8, 192)</td>
<td>0</td>
<td>otn3_1[0][0]</td>
</tr>
<tr>
<td>average_pooling2d_1</td>
<td>(None, 5, 3, 192)</td>
<td>0</td>
<td>concatenate_4[0][0]</td>
</tr>
<tr>
<td>dropout_3 (Dropout)</td>
<td>(None, 5, 3, 192)</td>
<td>0</td>
<td>average_pooling2d_1[0][0]</td>
</tr>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 2880)</td>
<td>0</td>
<td>dropout_3[0][0]</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 2)</td>
<td>5762</td>
<td>flatten_1[0][0]</td>
</tr>
</tbody>
</table>

Figure 31: Network parameters
References


