UPPSALA
UNIVERSITET

Pattern matching with neural networks for the PANDA at FAIR experiment

Jakob Andersson, Kerstin Ersson

Project in Computational Science: Report

January 2018

PROJECT REPORT

Institutionen fr informationsteknologi

Abstract

This paper investigates the feasibility of using machine learning techniques, specifically neural networks, to augment the algorithms used at the upcoming next-generation antiproton experiment PANDA at FAIR in Darmstadt, Germany. It will be one of the first particle accelerators where the data selection relies entirely on a software filter. The software in the experiment will have to handle data frequencies as high as 20 MHz, placing great requirements on the speed of the trigger. To combat the challenge, algorithms are being developed to extract information from the raw incoming detector signals. For this project, we have focused on how neural networks can be a part of these algorithms.

The specific kind of neural network that has been used in the project is a convolutional neural network. CNN:s are suitable for image recognition problems, or other problems where spatial relations are important, and the raw data from the part of the detector that is studied can be represented as images. As this report shows, machine learning techniques could definitely be of use for the PANDA at FAIR experiment, and we recommend investigating these methods more thoroughly.

Contents

Abbreviations	
PANDA	anti-Proton ANnihilations at DArmstadt
FAIR	Facility for Antiproton and Ion Research
STT	Straw Tube Tracker
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
TF	TensorFlow

1 Introduction

The PANDA project (anti-Proton ANnihilations at DArmstadt) is an experiment at FAIR (Facility for Antiproton and Ion Research), which is currently under construction in Darmstadt, Germany. The data selection at PANDA will be based solely on software, which makes this one of the first accelerator-based experiments of its kind.

The Uppsala PANDA team is particularily interested in hyperon reactions, and their decay vertices. The aim is to investigate the strong interaction between the quarks which make up the hyperons, as it is responsible for about 99% of the visible mass in the universe. When decaying, these particles have a relatively long lifetime, being able to travel up to a few meters. Today, there is no solution to recreate their tracks, due to the decay vertices being displaced from the interaction point. It is a challenge this project will try to help solving.

Since the raw data flow is very large, up to 200 GB/s, the need to quickly filter out the interesting data is high. One of the filter algorithms uses a pattern matching technique within a subdetector of PANDA.

The aim of this project is to extract information from the particle hit patterns of the subdetector, and using that information to investigate how the filter algorithm can be improved using machine learning, i.e. neural networks.

1.1 Problem formulation

To limit the scope of the project, we will be following two specific paths. Can we, using neural networks, identify

1. Particle types in single tracks?
2. The number of tracks in a collision event?

We will also consider the computational aspect of our implementations, as this is an important constraint to the PANDA project.

2 Background

This section will describe the hadron physics the PANDA experiment focuses on, how neural networks work and the frameworks used for this project.

2.1 Hadron physics

Hadrons are composite particles made of quarks. When a particle collides with its antimatter counterpart, they annihilate each other and generate a blast of energy. By studying the particle-anti particle annihilation process, one can gain knowledge about the strong interaction between the quarks. This is of interest as the strong interaction is responsible for about 99 % of the visible mass in the universe.[1] Figure 1 shows an example of a proton-antiproton annihilation decay vertice. Here, a proton is colliding with an antiproton and decaying in several steps. The dotted line shows two particles with no charge, which will not trigger a hit in the PANDA detector.

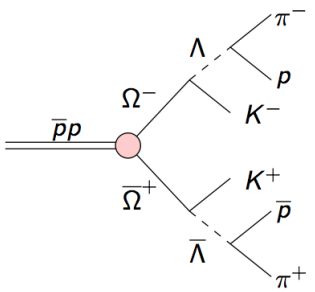

Figure 1: Example of a proton-antiproton annihilation decay process

2.2 The PANDA project and detector

The aim of the PANDA project is to investigate the weak and strong nuclear force, the exotic states and the structure of Hadrons.[2] The detector itself is currently under construction, but the proposed build-up consists of the target spectrometer and the forward spectrometer, which together allow for full angle coverage.[3] In Figure 2 an example of how the detector could look when finished is shown.

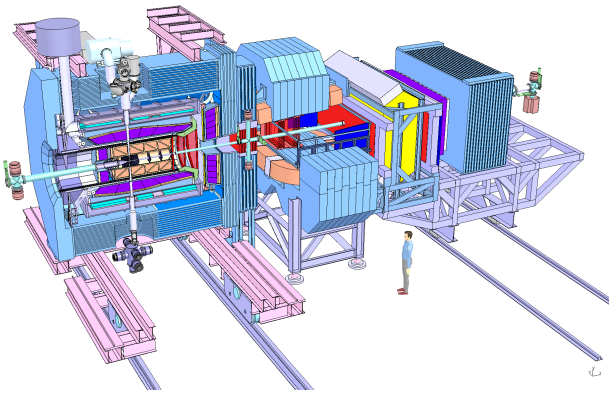

Figure 2: A design example of the proposed PANDA detector

2.2.1 The Straw Tube Tracker

The particle tracking system consists of four components. In this report, we will focus on a part of the Central Tracker in the target spectrometer, the Straw Tube Tracker. In the proposed setup, the STT is build up by 4224 gas filled tubes which have wires along the center axis. In the gas filled area, an electric field is generated by applying a high voltage to the wire in the tube. When a charged particle passes a tube, it will ionize the gas and charged particles in the gas will drift towards the conducting wire. This will cause the STT to register a so-called hit signal. By combining these hits, one can reconstruct the trajectory of a passing particle.[4]

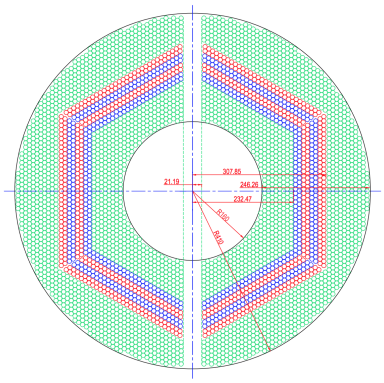

Figure 3: A cross section of the STT layout

The straw tubes in the PANDA STT are arranged in a barrel-like shape.

Some of the tubes are slightly inclined, shown in Figure 3 in red and blue, to better determine the longitudinal position of a particle. However, in this project, we will only be focusing on the xy-plane positions.

2.3 Classification

Classification is a method to extract information from data sets. This is done by dividing the data into categories based on some features. The idea is to derive a model which can perform the sorting process by training it on data objects where the category, or label, is known. The model should then be able to classify unlabeled data with sufficient accuracy. There are many different models that are used for classification, e.g. neural networks.[5]

2.4 Artificial Neural Networks

Machine learning is a field in computer science aiming to imitate the human learning process. Artificial neural networks, or just neural networks, is a kind of machine learning technique where the structure of the human brain is the inspiration.

The artificial neural network (ANN) is a network built of a number of interconnected neurons. The neurons are simple processing units that change their internal state, or activation, based on the current input and produces an output that depends on both the input and current activation. Such a neuron and its biological counterpart is shown in Figure 4. The ANN is constructed by having a large number of these neurons working in parallel and connecting some neurons to others through weighted connections, creating a weighted and directed network of different layers. It is by adjusting these weighted connections and the internal activations of the neurons the ANN can be improved, or trained. Usually the network cycle through a set of training data sufficiently many times, until the weights have been adjusted enough to produce the desired output.[6]

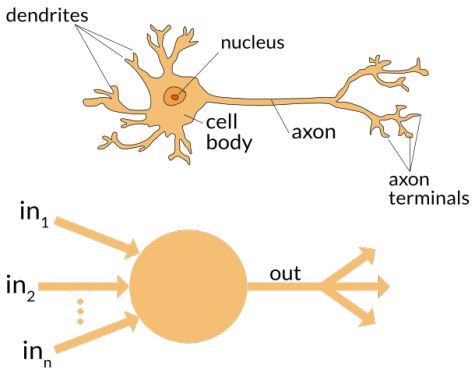

Figure 4: A comparison between a biological and an artificial neuron

A common way of learning for a neural network is the process of back propagation. This is a method of dynamic adjustment based on providing feedback to the network, initiated from a difference between the desired output and the current output. The weights of the interconnected neurons are adjusted depending on the degree they contribute to the error and this process is repeated in cycles until the network achieves a desired accuracy of classification. [6]

2.4.1 Convolutional Neural Networks

For image recognition problems, convolutional neural networks, or CNN:s, are a common choice. A common choice for the structure of a CNN consists of alternating layers of convolutional and max pooling layers, which are then combined in a dense layer.

The convolutional layers work as a feature extractor, or a filter, and extract some sort of characteristic from the input data. Usually, one convolutional layer has several filters which are all applied in the same step, but extract different features. The size of the filter, or the *kernel*, depends on what the size of a specific feature is expected to be.[9]

After the convolution is done, some kind of non-linearity is often applied. The most common choices include the sigmoid function and the ReLU function, used in our project. ReLU is defined as $f(x) = max(0, x)$, thresholding the input at zero. ReLU has become a common choice, as it can help to increase the convergence of the gradient descent optimization method.[10]

When a feature has been extracted from the image, we can reduce the spacial

size of the image. This will keep the information of which features are present and their relative positions, but in a lower resolution. This process is called subsampling and is done by a pooling layer. In this case we are using a max pooling layer, which is a common choice for convolutional neural networks. This specific king of pooling means that from the pooling kernel, only the pixel with the largest intensity value will be transferred into the subsampled image. Looking at Figure 5, we can see that for each similarly colored two-by-two square, only the highest value has been transferred to the subsampled square.[9]

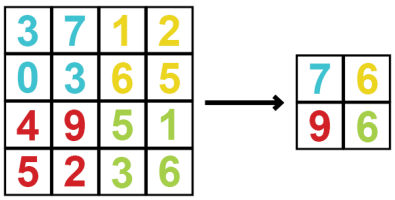

Figure 5: An example of max pooling

2.4.2 Training a network

As stated in the ANN section, CNN:s are trained using back propagation. In order to adjust the weights correctly, we need a sufficiently large set of training data. There is no clear formula for how big this data set should be, but one aspect that is important to consider is variance between classes. If the disparity within a class is big, the number of training data objects should be larger.

During the training process, for each *step*, one data object is run through the model and the weights are adjusted. When all training data has passed through the network once, one *epoch* is completed. The number of steps and epochs are important parts of the training process, as too few or too many can lead to under- or overfitting.[10]

A model is overfitted if it is too well adapted to the training data, but does not perform well in the general case. One possible cause of overfitting is if the classes in the training data are unbalanced. Underfitting occurs when the

chosen model does not fit well with the data and causes "overgeneralisation" by the model. One way to combat overfitting in neural networks is the use of so-called regularization. [10]

2.4.3 Loss function

An important part of the design of the network model, is choosing the loss function. The loss function represents the price paid for inaccuracy in predictions made by the neural network. By minimizing the loss function during the training process the error of the network also will be minimized. For classification problems, one of the most popular choices for loss function is the softmax cross entropy functions, defined as:

$$H_{y'}(y) = -\sum_i y_i' log(y_i),$$

where y is the predicted probability function and y' is the true distribution.[14]

Gradient descent is a common optimization method for minimizing the loss function in machine learning algorithms. The idea is to follow the steepest descent of a function F(X), which will be proportional to the negative gradient, $-\nabla F(X)$ to find minima of the function. This is illustrated in Figure 6 where the red line shows the direction in each iteration step for the gradient descent algorithm, towards the minimum in the center.[10]

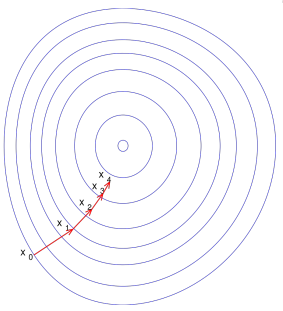

Figure 6: Illustration of the gradient descent method

3 Hardware and software

3.1 Hardware

Since the computational aspect of the project is of importance to PANDA, it is important to know the hardware that was used in the evaluation process. The training and evaluation of the neural network model has been done on a Windows 10 computer using a quad-core CPU at 3.4 GHz.

Comparing the results from home user-level of hardware to the kind of hardware that should be available to a particle detector could give us an inkling of how effective this kind of CNN could be. Let us also note that the final experimental setup will be generating events at frequencies of 2 or 20 MHz, which is relevant in the sense of data points to process per second.[13]

3.2 Software

3.2.1 TensorFlow

TensorFlow (TF) is an open source API developed by Google mainly for Machine Learning and Deep Learning, but it is also applicable for other numerical computations. Google uses TF in some of its commercial products, such as their speech recognition API and Gmail, but it is also used for research. The framework can be used both as backend, with C++, and frontend with Python.[8]

One of the advantages of using TF are the many built in functions. We are using the high-level API Estimators, which can be customized if needed. For this project, we have built our own estimator with a custom model. This model is utilizing the TF built in functions *tf.layers* and *tf.losses*, defining the structure and loss function of our neural network.

Loss function
We have used the softmax cross entropy function in the *tf.losses* module. The parameters available for the loss function include weights, scope, reduction and smoothing. Weights can be used to bias the network toward a certain class, e.g. to combat overfitting. The optimization method is also a parameter for the network.

Layers

The layers in the *tf.layers* module include the standard CNN layers convolution and pooling. We work with the 2D convolution and max pooling layers for this project. Some of the parameters available for the convolutional layer include number of filters, kernel size, strides, padding and activation function. For the max pooling layer the available parameters are kernel size and strides.

The padding determines how the filter will function at the edges of the image. To keep the spatial size, one can use zero padding around the borders, adding one or more rows/columns of zeros. The number of pixels between the application of filters is determined by the strides parameter. Together, the parameters number of filters, padding, and strides determine the size of the output of the layer.

Other commonly used layers in CNN:s are the *flatten* and *dropout* layers. Dropout layers are used to prevent overfitting, where the user provides a rate between 0 and 1 at which the model will randomly drop neurons. The flatten layer reshapes the data and makes it ready to use in the prediction layer.

Other parameters
The batch size of the network is the number of data points passed through the network in each propagation. After one batch is processed, the weights are updated.

The learning rate is a measure of how much the model is inclined to abandon the belief that a certain feature is most predominant, and instead choose another feature. Both too high and too low learning rates can make the training process longer.

Parameters for the training process, such as the number of steps for the training, are also available for user modification.

3.2.2 PandaRoot

The software used to simulate the data in this project is PandaRoot, developed particularly for the PANDA experiment. PandaRoot is written in C++ and can simulate many variations of particle collision events as how they should appear in the real detector. We have been using a class of PandaRoot called PndPatterMatcher, where we can simulate the annihilation events which are in our interests and store the resulting STT hits.[2]

3.2.3 MATLAB

MATLAB is a computing environment used for implementation of algorithms, numerical computations, and matrix manipulations. It can also be used to interface with other programming languages. Some image analysis is also possible in MATLAB and that is what it mainly has been used for in this project.[15]

4 The data formatting process

4.1 Simulating the data

Since the PANDA detector is not yet built, we have used simulated data to train and evaluate the network. From the raw simulated data of PandaRoot it is possible to extract a 1D array containing the STT tube IDs which are registering hits for a certain event.

Apart from using the raw simulated data, we have also made an effort to extract so called ideal tracks to another dataset. This dataset contains only the events for which there is only one particle present in the STT. For these events, the particle type is easier to be automatically identified and the dataset is required for one of the paths of the project.

In total, the simulations have generated over 800 000 events. Out of these, different subsets have been used to train each neural network model, ranging in size from a few thousand up to fifty thousand for the largest ones.

4.2 Array to pixel image

The raw simulation data is in 1D, which theoretically could be used directly in a neural network. However, as the trajectories can be projected to a cross section in the XY-plane of the STT, it seems intuitive that this data could be represented by a 2D pixel image. The STT tubes are, however, placed in a hexagonal barrel shape, where the coordinates of a tube in the XY-plane cannot be correlated to a precise pixel coordinate in an image. [4]

To subvert this problem, we needed a way to translate the known XY-plane coordinates of the tubes to pixel intensities. The solution was implemented in Matlab and is described by the equation

$$\sum_{i=1}^{4} p_i = \frac{I}{\sqrt{(x_i - x_{pos})^2 + (y_i - y_{pos})^2}}$$

where $\sum_{i=1}^{4} p_i$ are the four closest pixels to a coordinate, I is the intensity level of a white pixel, x_{pos} and y_{pos} are the XY-plane coordinates of the tube and finally x_i and y_i are the pixel coordinates for one of the four closest pixels. This is used to spread the intensity across the closest pixels with a weight towards the closest ones.

To save memory, we chose to use rather small images, 80 x 80 pixels in greyscale, so called 1-channel images. This choice of size is not accidental since it is representative for the spatial dimensions of the STT, in which the tube width is about the size of a centimeter and the total width is about 80 centimeters. The image format that was chosen is JPEG, a common file format which has many built in functions in TF.

4.3 Labeling training and evaluation data

4.3.1 Data for identifying the particle type of a single track

For the first part of our project, we were able to label the images by extracting information from the simulations. Since there was only a single particle present in the STT for these events, the simulation program could automatically provide labels for each event.

4.3.2 Data for identifying number of tracks in an event

On the other hand, labeling of training data for the second path of the project could not be done automatically. Since the information of which particle caused what STT hit isn't directly available, a specific particle identity could not be related to a specific track in the simulation events. Because of this, the labeling of this data had to be done manually to get any usable data at all. We examined the resulting images, of which examples are shown in Figure 7, and tried to accurately identify the number of tracks based on our perception o the image. In the example the left-most image would be labeled as one track, the middle one as two tracks and the right-most as four tracks.

Figure 7: Three examples of collision event images

4.4 Class distribution within datasets

The simulations of the PANDA detector have used a realistic angular distribution of the particles in the collision events. This means that the particles have a preferred angular direction, and thus some particles might not produce hits in the STT, which in turn affects the class distribution.

After extracting data from the simulations we were able to construct datasets for use in the neural network models. Table 1 shows the class distribution for the dataset used in the particle classification path of the project.

Table 1: Distribution of data for the single track dataset

Parameter	**Performance**
No particles	3.4 %
Proton	26.2 %
Anti-proton	0 %
Pion	0.3 %
Anti-pion	70.1 %

For the second path of the project another dataset was used, and this is the one that was labeled manually. In Table 2 the class balance for this dataset is shown to be slightly less uneven compared to that, shown in Table 1.

Table 2: Distribution of data for the number of tracks dataset

Parameter	**Performance**
No tracks	22.3 %
One track	53.4 %
Two tracks	17.7 %
Three tracks	2.2%
Four or more tracks	4.4 %

5 The resulting neural network

Since we are using 2D images as data for the network, a CNN is our choice of neural network variant, as they are superior in that field. Also, CNN:s are usually simple to use, as they have fewer parameters than other kinds of ANN:s and they are usually easy to train.

We chose a relatively simple structure of our CNN, with five hidden layers. Two convolutional layers are alternated with two max pooling layers, as is seen in Figure 8. The final layer before the output is a dense and fully connected layer, condensing the individual weights of the neurons in the network into network predictions.

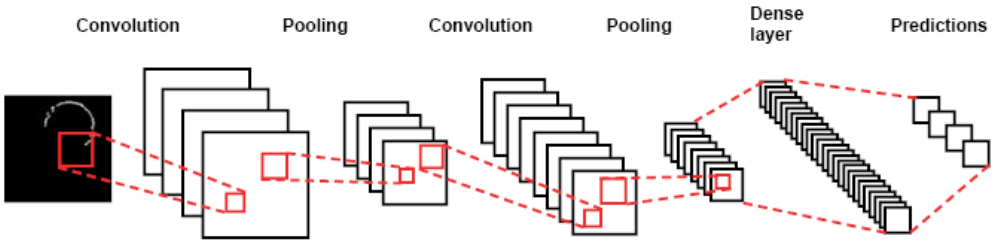

Figure 8: The setup of the neural network

5.1 Choice of network parameters

The input data of the network is given in batches. In order for the network to use the data for training, it has to be reshaped into tensor format. In our case, we have 80 x 80 pixel images in greyscale, i.e. with one color channel. The resulting tensor passed through the network changes form during the

process, but the initial shape is $[batch\_size, 80, 80, 1]$. Our choice of batch size is 100 for this model, as the network usually trains faster for smaller batch sizes.

There is no one right answer to how to choose the network parameters. A common method when finding good parameters is to simply try different values and compare the results. As time was a serious constraint in our project, we have not been able to try a large number of parameters. The parameters used in the model that we evaluated are listed in Tables 3, 4, 5 and 6. Comments on the choices and thoughts of improvement are listed in the discussion section of this report.

Table 3: First convolutional layer

Input data shape	**[100, 80, 80, 1]**
Number of filters	32
Kernel size	5x5
Padding	None
Activation	ReLu
Resulting data shape	**[100, 80, 80, 32]**

Table 4: First pooling layer

Input data shape	**[100, 80, 80, 32]**
Kernel size	4x4
Strides	4 (no overlapping)
Resulting data shape	**[100, 20, 20, 32]**

Table 5: Second convolutional layer

Input data shape	**[100, 20, 20, 32]**
Number of filters	128
Kernel size	5x5
Padding	None
Activation	ReLu
Resulting data shape	**[100, 20, 20, 128]**

Table 6: Second pooling layer

Input data shape	**[100, 20, 20, 128]**
Kernel size	2x2
Strides	2 (no overlapping)
Resulting data shape	**[100, 10, 10, 128]**

5.2 The parameters for the training process

The loss function used in our CNN is softmax cross entropy, a common choice for CNN:s. We are not using any of the modification parameters available in TensorFlow, but will discuss possible improvements regarding the loss function in the discussion section of the report. We settled for a learning rate of 0.001, which was sufficient in our case. For the optimization method, we used gradient descent.

The number of steps in the training process was a parameter that we investigated when we evaluated the accuracy of the network. The results are presented in the network performance section of the report.

Due to time limitation, we did not modify the network parameters to be customized to the two paths we chose. As the images and the features are of the same size for both approaches, this might not be necessary, but it could be worth looking into.

6 Model accuracy and performance

6.1 The single track classification model

Here we present some results for the model trained to match single track images from the STT to a certain particle. In the first graph, Figure 9, the accuracy of the network is plotted over its training period. The dataset used for this specific plot contain 10000 STT collision event images. In the graph we are showing the accuracy for the network, both when it is evaluating data from a test dataset and when it is evaluating on its own training dataset. The discrepancy between the two curves could be a result of overfitting, which is when a model has become too good at classifying data in its training dataset but is not as good in the general case.

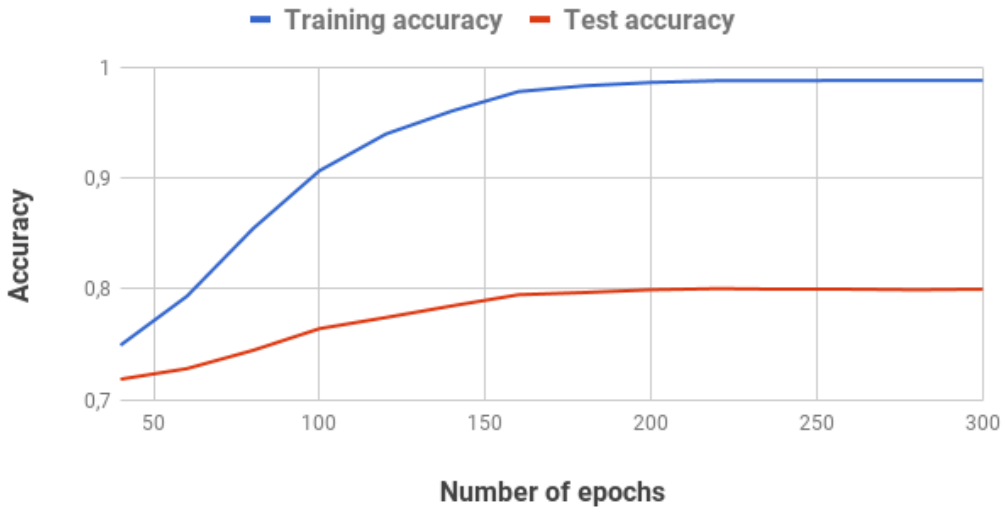

Figure 9: Accuracy vs. epochs for the single track classification model

Figure 10 shows the loss function projected over the course of training one of the convolutional network models. It shows the same kind of behavior as in Figure 9, leveling out after a certain number of training steps.

Figure 10: A graph of the loss function vs training time for a neural network model

6.2 The number of tracks classification model

Since there was no way for us to properly generate a large dataset of images to pursue this kind of network model the results are inconclusive. The dataset we did use was of 2000 collision events and all of it had been labeled manually. When trained on this data the resulting neural network model reached as high as 70 %.

6.3 Network performance

It is interesting to note some of the performance related results for the neural network models. The results are an average gathered from several models of the same type of neural network which all were run on the same computer (as described in Section 3). The performance results are shown in Table 7 and are of relevance when discussing the use of the neural network model for the real PANDA detector.

Table 7: Some of the performance results

Parameter	**Performance**
Size of one image	0.58 kB
Size of the model	$\sim$ 130 MB
Evaluation time per data point	2.52 ± 0.37 ms

7 Discussion

7.1 Accuracy results

When the project first started we had hoped to reach an accuracy of more than 90 percent for some configuration of our neural network model. However, within the scope of this project, we did not manage to meet these expectations for either of our chosen paths.

For online classification, a slightly lower accuracy can be acceptable if the solution meets the computational requirements. However, an accuracy above 90 percent is a necessary minimum for both an online and an offline model.

Regarding the accuracy of the number of tracks model we believe that the low accuracy, around 70 % as highest, can be partly attributed to the small size of the dataset. Since the labeling had to be done manually we couldn't produce enough data, or data of assured quality, to properly train the neural network model. The tests we did does however show promise for the setup.

We do not believe that we have reached the best possible accuracy for a CNN with images as input data, we have only created a first draft of such a model. In the following sections we discuss some of the ways in which we believe that our model can be improved.

7.2 Network configuration

Due to the time limitations of this project, we were not able to thoroughly investigate the ideal parameters of our network. As almost every parameter affects how the network trains and, following that, how good it becomes at classifying its input it is something that logically should have an effect on the result. We did look into some of the parameters of the training process, but the structure of the network as well as the parameters for the layers and the loss function could use more work.

As noted in the result section, we experienced overfitting with our network. One way to counter this could be to bias the network toward the less represented classes, by changing how the weight parameters are adjusted by the loss function.

7.3 Training data

A likely cause of the problem with overfitting was the dataset used for training. It had an unbalanced class distribution and to get better training data, a few different things could be done. There are settings in PandaRoot, allowing for simulations which would result in a better spread between different kinds of collision events and in turn different kind of classes. Also, getting more training data would allow for some preprocessing methods. One example is leaving out some of the data points of the more represented classes.

Especially in the case of identifying the number of particle tracks in an event, more training data would be useful. The distinction between classes is lower in this approach, and the variety within classes is also higher.

7.4 The computational aspect

As previously mentioned, the performance of any software filter will be of great importance at the PANDA at FAIR experiment. The most relevant factor for this is the evaluation time of the network, how long it takes to classify a single data point. Comparing the evaluation time we achieved in the project, ~ 2.52 ms, to the time between events in the detector, ~ 0.5 μs, we can see that it is about three magnitudes off. This is quite a lot but there are a number of parameters that can be improved to gain a much higher performance.

Considering that our tests have been run on a personal computer, using a single CPU, gives us reason to believe that the PANDA detector will perform better. This is due to the fact that the finished detector will likely have access to more advanced hardware. For instance, it is possible to spread the calculations across several nodes, each node only taking on part of the load, making the overall computation much faster. Using GPUs designed for neural networks or other kinds of dedicated hardware are also options with the potential of greatly increasing the evaluation speed. [13]

It does take some time to train a network, anything from an hour to a whole day, depending on the complexity and setup. However, this has to be done only once per neural network model and does not really affect how it performs at runtime. The network model's size is not a problem as well, from the results in Section 6.3 we gather that individual event images are on average 0.58 kB with the complete model averaging at around 130 MB. These are quite small numbers with respect to modern memory hardware being able to achieve transfer rates of over 20 GB/s. [16]

8 Conclusions

We would recommend further investigations into the use of neural networks for the PANDA project, as we believe that it has great potentials in many aspects.

The specific work we did in this project, implementing a CNN for identification of particle type of a single track and number of tracks in an event, show promise with regards to computational performance. Concerning the accuracy, the network did not perform as well as we would have hoped. However, we believe that with some improvements of the training data set

and network parameters we could improve the accuracy sufficiently.

8.1 Outlook

The future development we would recommend includes

- Improving the training data by several means: simulate for better balance, using preprocessing, get more data for the number of track identification.
- Customizing the loss function if well balanced training can not be achieved.
- Tuning and customizing the parameters of the network, regarding structure, layers, filters etc.
- Optimizing code by e.g. porting everything to C++ and using other optimization methods.

References

[1] Claude Amsler, *Proton-antiproton annihilation and meson spectroscopy with the Crystal Barrel*, Rev. Mod. Phys., Vol. 70, No. 4, October 1998

[2] The PANDA Experiment Website, `https://panda.gsi.de/`, Accessed on December 10th 2017

[3] Overview of the PANDA detector design, `https://panda.gsi.de/article/panda-detector-overview`, Accessed on December 10th 2017

[4] The Straw Tube Tracker, `https://panda.gsi.de/article/straw-tube-tracker`, Accessed on December 10th 2017

[5] Jiawei Han, Jian Pei, Micheline Kamber, *Data mining: Concepts and techniques*, Third Edition, 2012, pp. 18

[6] George F. Hepner, *Artificial Neural Network Classification Using a Minimal Training Set: Comparison to Conventional Supervised Classification*, PHOTOGRAMMETRIC ENGINEERING AND REMOTE SENSING, Vol. 56, No. 4, April 1990, pp. 469-473.

[7] Bottou Léon (2010), *Large-Scale Machine Learning with Stochastic Gradient Descent*. In: Lechevallier Y., Saporta G. (eds) Proceedings of COMPSTAT'2010. Physica-Verlag HD

[8] TensorFlow main website, `https://www.tensorflow.org/`, Accessed on December 8th 2017

[9] LeCun, Yann, Bengio, Yoshua (1997), *Convolutional Networks for Images, Speech, and Time-Series*. Retrieved from `https://www.researchgate.net/publication/2453996_Convolutional_Networks_for_Images_Speech_and_Time-Series`.

[10] Karpathy, Andrei (2017), Lecture notes from *Convolutional Neural Networks for Visual Recognition* at Stanford University. Retrieved from `http://cs231n.github.io/`.

[11] van der Aalst, Rubin, Verbeek, van Dongen, Kindler, Günther (2008), *Process mining: a two-step approach to balance*

between underfitting and overfitting, https://link.springer.com/content/pdf/10.1007%2Fs10270-008-0106-z.pdf

[12] TensorFlow losses module, https://www.tensorflow.org/versions/r1.5/api_docs/python/tf/losses Accessed on January 13th 2018

[13] Michael Papenbrock, project supervisor

[14] TensorFlow Layers guide, https://www.tensorflow.org/get_started/mnist/beginners Accessed on January 14th 2018

[15] Matlab documentation, https://se.mathworks.com/products/matlab.html, accessed on January 14th 2018.

[16] RAM transer rates, https://www.transcend-info.com/Support/FAQ-292, accessed on January 21st 2018.