Simulating background noise of radio neutrino detectors with a Generative Adversarial Network (GAN)

Bachelorarbeit aus der Physik

Vorgelegt von
Simon Hillmann
Abgabedatum: 27. Juli 2021

Erlangen Center of Astroparticle Physics
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuerin: Prof. Dr. Anna Nelles

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.
Die Richtlinien des Lehrstuhls für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Erlangen, den 27. Juli 2021

Abstract

This thesis explores the efficacy of using a Wasserstein generative adversarial network to simulate background noise of radio neutrino detectors in order to improve on the current method of adding real noise to simulated neutrino events. To do so different network architectures were tested and their performance was compared.
The sample traces generated by the best performing network show good agreement with some of the characteristics of real noise traces, but are not yet capable of fully reproducing the detector noise accurately.

Contents

1 Introduction

This chapter contains a motivation for the thesis. It will also introduce the concept of detecting neutrinos with the radio technique and give an overview of current projects utilizing this technique.

1.1 Motivation

The main goal of this thesis is to construct and to train a Wasserstein-Generative-Adversarial-Network, WGAN for short, to simulate realistic radio noise as expected in the antennas used at RNO-G or ARIANNA.
This noise can then be used in simulating neutrino events. This is important for the analysis of the actual data, where realistic simulations are necessary. At the moment noise is added to the simulation, by reading in real data from the detector only containing noise. This process is rather impractical because it requires gigabytes of noise data to work optimally. Using less data would result in adding the exact same noise to different simulations. Having to download and store these files to do analyses is inefficient. Since radio noise is notoriously hard to simulate this is the most accurate option available right now.
Using WGANs in physics simulation has shown promising results in other applications. Therefore this thesis is looking into using this technique to improve the process of generating realistic noisy simulations.

1.2 Radio Detection of Neutrinos

The radio detection of neutrinos is a recently developed technique aiming to detect Ultra High Energetic (UHE) neutrinos. It utilizes the Askaryan-effect, which is depicted in Figure 1. A neutrino can interact while traveling through a dense medium and produce a shower of secondary particles. When this particle shower propagates through the medium the shower develops a net negative charge via Compton-up-scattering of medium neutrinos into the shower or by shower positrons annihilating with electrons of the medium. The negatively charged shower then emits electromagnetic radio waves on the Cherenkov cone because of the relativistically moving electric charge. This radiation can then be detected and the information can be used to reconstruct the neutrino event.
The radio technique has an energy threshold of $\mathcal{O}(10^{16}$ eV) and is therefore specifically geared towards detecting UHE neutrinos. The longer in-ice attenuation length in the order of ~ 1 km of radio signals, compared to the attenuation length of optical signals of around 100 m, allows a radio detector to cover a larger effective volume, in which neutrino events can be detected. The radio technique enables us to see neutrinos past

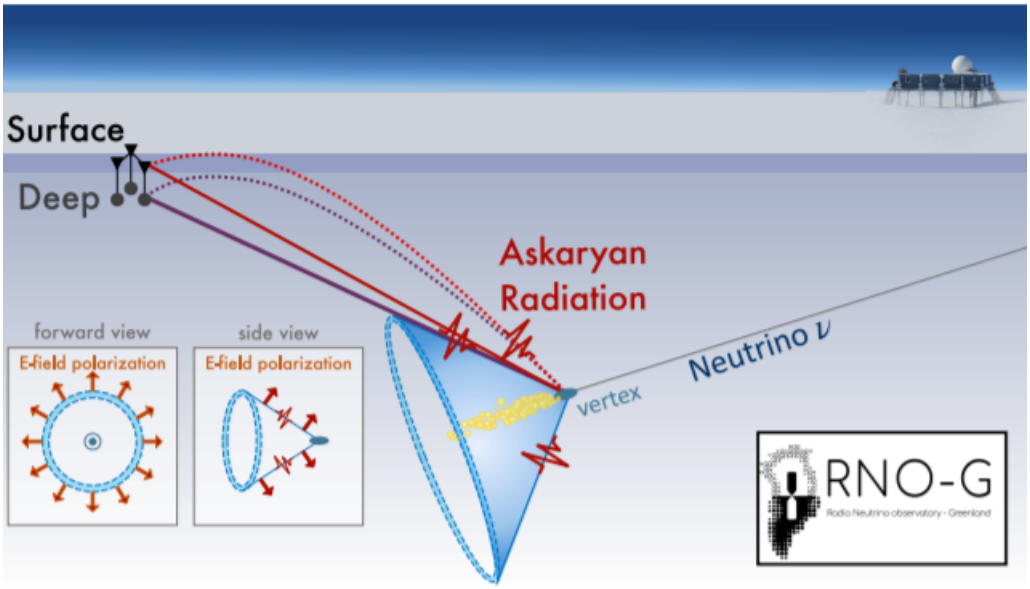

Figure 1: Schematic depiction of the Askaryan-effect. Source: [1] Page 11

the current energy limitations of optical detectors.

1.3 Experiments for Radio Detection

The first experiment utilizing the radio technique was the Radio Ice Cherenkov Experiment (RICE). RICE was located at South Pole station and operated from 1999-2010. RICE gave the first insight into operating radio detectors at remote locations and at depths of up to 200 metres. It also set the first limits on the diffuse neutrino flux from a radio detector [2].

Considered to be the direct successor of RICE is the Askaryan Radio Array (ARA), which has been operating since 2010. In contrast to RICE, which used ice-filled bore holes of the optical detectors AMANDA and IceCube [3], ARA stations use specially dedicated dry holes. A station consists of 4 receiver strings each equipped with 2 antennas to detect vertically polarized radiation (VPol) and 2 antennas for horizontal polarization (HPol) [3]. ARA also pioneered the phased array trigger technique in its last station, which increased the effective volume by a factor of ≈ 2 [4].

Also, in 2010 the construction of the Antarctic Ross Ice shelf ANtenna Neutrino Array (ARIANNA) began [5]. ARIANNA uses a surface array concept, where high gain Log-Periodic Dipole Antennas (LPDAs), capable of detecting broadband radio frequencies both vertically and horizontally polarized because of the antennas geometric arrangement and not being restricted in size by the bore hole geometry, are buried just underneath the snow surface. Each station consists of four LPDAs. ARIANNA also

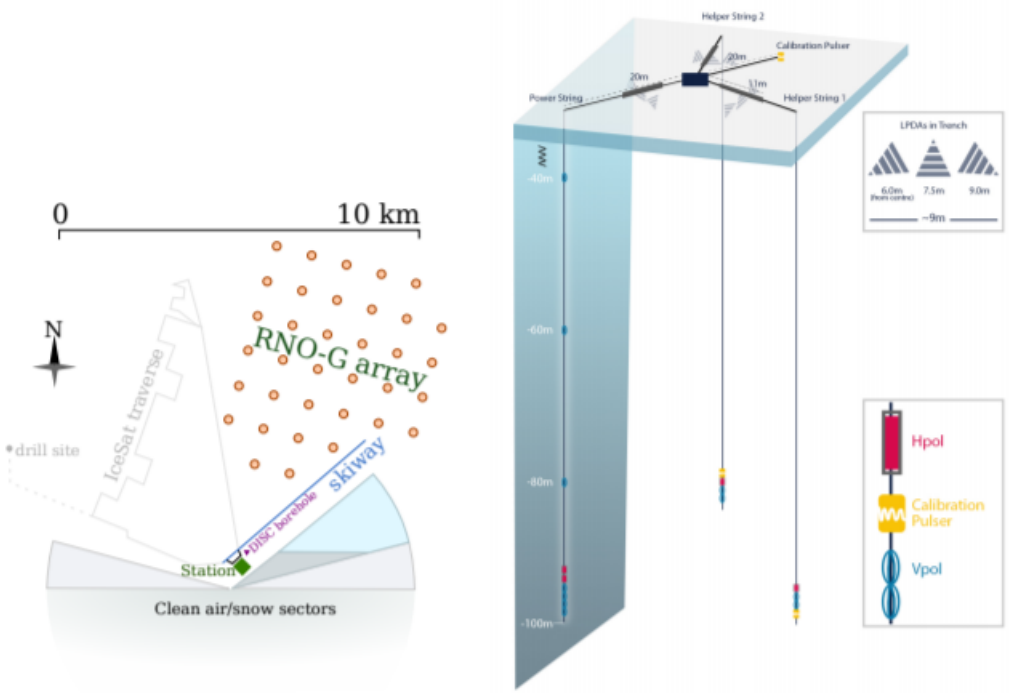

Figure 2: Left: Planned Layout of the RNO-G station positions; Right: Planned setup of a RNO-G station; Source: [1] Page 13

utilizes a geographical effect due to being built on an ice-shelf. The ice-water interface at the bottom of the ice-shelf acts as an almost perfect mirror for radio waves. This allows downward going neutrino signals to be reflected and then travel back up to the antennas.

1.4 Radio Neutrino Observatory in Greenland

The Radio Neutrino Observatory in Greenland (RNO-G) is a neutrino detector currently under construction at Summit Station in Greenland. Once finished, RNO-G will be the first production-scale neutrino detector utilizing the radio-technique. The main goal of RNO-G is to make the first observation of neutrinos with energies above 30PeV and up to several EeV emitted directly by astrophysical sources and emitted by interactions of ultra high energy cosmic rays with photons of the cosmic microwave background [6, 7].

The planned setup of a station for RNO-G is depicted in Figure 2. Combining the knowledge gained from all prior radio neutrino experiments the station will consist of a combination of three strings equipped with a combination of HPol- and VPol-antennas, placed up to 100 meters deep in the ice, utilizing the phased array trigger and also surface antennas.

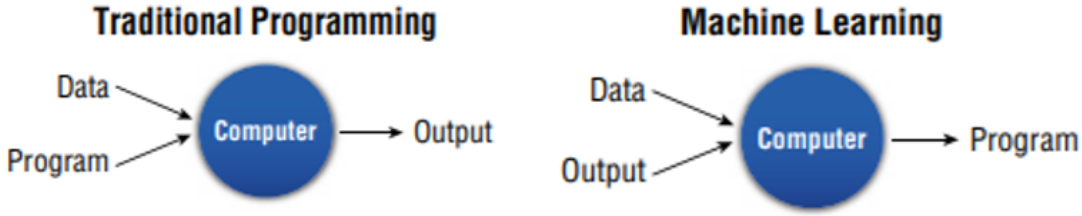

Figure 3: Left: schematic diagram of traditional programs; Right: schematic diagram of machine learning programs; Source: [8]

Since RNO-G is currently under construction there is no data from the detector available at the moment. Therefore training of the generative network will be done with data from ARIANNA as a proof of concept. One can train the network analogously with RNO-G data once it is available.

2 Machine Learning & GANs

This section focuses on introducing a theoretical background on machine learning and neural networks in general and on the special case of Generative Adversarial Networks and their application in a physics context.

2.1 The concept of machine learning

Machine Learning (ML) is a rather young and very interesting topic of computer science. Machine learning fundamentally changed the way one can look at solving problems computationally. The traditional approach to programming would be, to write a program and feed it into a computer together with some data and receive an output, as it is depicted in Figure 3 (Left). In this case all the "hard" work is done by the programmer. The problem is already solved by writing the program and the computer only does the calculations in the end.

ML introduces a new approach. Instead of solving a problem by writing a program, through ML one can write a program, which solves the problem for you. This process is also depicted in Figure 3 (Right). You take some data and outputs and feed them into the computer. The computer will then form a program, also called a model. For example one could try and solve an accounting problem. First you collect data in the form of a detailed sales record, which lists how many copies of a certain item you sold in the past years. This data acts as both the data and the output in Figure 3. The computer then learns rules about the data through a ML-algorithm and can then predict for example which item will be the most popular one in the next year [8].

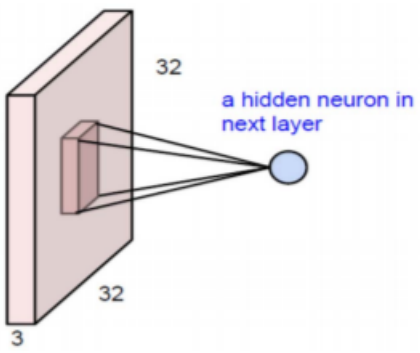

Figure 4: One region of the input beeing mapped onto 1 neuron in a convolutional layer. Source: [9] Page 2

2.2 Neural Networks

One subset of machine learning are Neural Networks (NNs). It is an advanced field in ML combining multiple ML-algorithms into a network to solve complex problems.

NNs are built to simulate a human brain by combining nodes, which are also called neurons, into different layers. The layers are connected to each other. The first layer is called the input layer, then follow a number of hidden layers and then the last layer, which is called the output layer. The most basic layer structure would be the fully connected layer. In a fully connected layer each node in the layer is connected to all nodes of the prior layer. The output of the node is then calculated via

$$a_{out} = \text{activation}\left(\sum_{i=1}^{N} a_i \cdot w_i + b_i\right) \tag{1}$$

where a_{out} is the output value of the node, N is the number of nodes in the prior layer, a_i are the values of the nodes of the prior layer, w_i are the weights for the node and b_i are the biases of the node. Activation stands for the so called activation function, which acts on the sum. Typical activation functions are for example the identity function or the Rectified Linear Unit function (ReLU), which is defined as

$$\text{ReLU}(x) = \max(0, x). \tag{2}$$

The weights and biases of each node in each layer are then optimized through training to yield the desired output. Other layer structures that are designed to perform specific mathematical operations are also possible.

For example one of these possible mathematical operations is convolution. In a convolutional layer the output neurons are not connected to all input neurons, but only to a small region corresponding to this neuron, which is depicted in Figure 4. This mapping

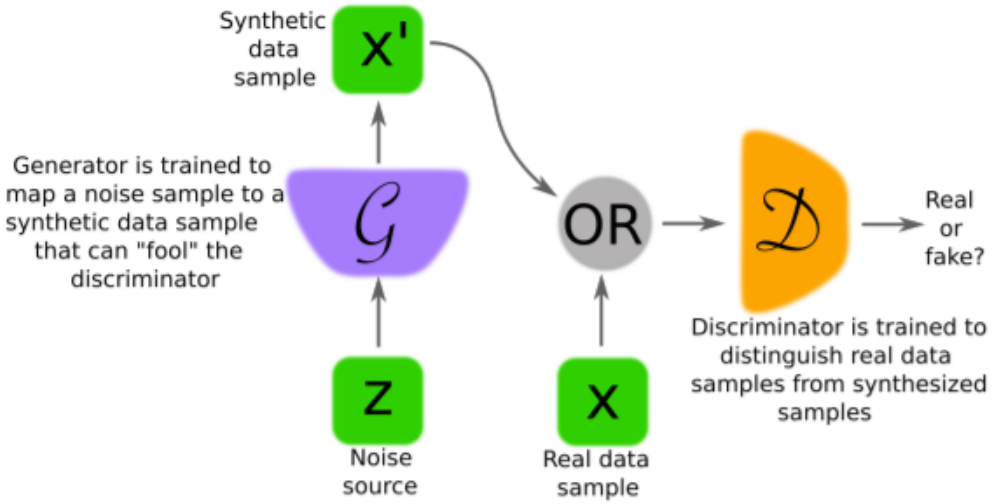

Figure 5: Interaction of the discriminator $\mathcal{D}$ and the generator $\mathcal{G}$. Source: [11] Page 2

of a small region of the input to one output allows the layer to detect features in the input regardless of its position, hence the name convolution [9]. These mappings work like classic filters in image processing and can be trained to apply the correct filter to the input to improve the output of the array. Convolutional layers allow a NN to learn the input structure on a more local level, because the filters only depend on a small field of neighbouring neurons. So for example the output of a certain neuron depends strongly on input bin two and three but is not influenced by bin 200.

To train a NN one first has to choose a loss function. This loss function is a measure of how well the NN is performing. Choosing which loss function to use, depends on the goal of your NN. For example when solving a regression problem, like predicting the price of an apartment by knowing some limited information about it (e.g. size, number of rooms, etc.). In this case one would probably use the Mean Squared Error (MSE) as the loss function [10]. One would train the network by feeding it the information about some apartments and then compare the predicted price with the actual price. Then the MSE of this data sample would be calculated and then the weights and biases would be adjusted to try and minimize this loss function.

2.3 Generative Adversarial Networks (GAN)

One application of the NN concept are Generative Adversarial Networks (GANs). A GAN consists of two NNs competing against each other. One is trying to generate fake samples of a given distribution; this network is called the generator. The other network, called the discriminator, tries to distinguish between fake samples and actual data samples. This process is depicted in Figure 5. The generator gets fed a random

number sample and tries to map it to the desired distribution. The discriminator gets fed either actual data or the generated fake sample and has to decide whether it is real or fake. Both networks are trained simultaneously and in competition against one another. Important to note is, that the generator has no access to actual data; it only learns from the feedback of the discriminator. Through this process the generator implicitly captures the underlying distribution of the data, a high-dimensional function, which would be very difficult to construct otherwise [11].

Typically the networks are implemented in a multi-layer structure consisting of convolutional and/or fully connected layers [11]. Training these networks is done by first defining a value function, which depends on both the discriminator and the generator:

$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{data}(x)} \log(\mathcal{D}(x)) + \mathbb{E}_{p_g(x)} \log(\mathcal{D}(\mathcal{G}(x))) \tag{3}$$

with $\mathbb{E}$ being the expected value, $p_{data}(x)$ the probability distribution of the data and $p_g(x)$ the probability distribution of the generated sample. Then optimization is done by solving

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{D}, \mathcal{G}). \tag{4}$$

The training is then performed by fixing the parameters of one network while optimizing the parameters of the other network and vise versa. In [12], Goodfellow et al. show, that for a fixed generator a unique optimal discriminator exists, which has the form $\mathcal{D}^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$ and that for this optimal discriminator the optimal generator has the form $p_g(x) = p_{data}(x)$.

In theory this process should result in an optimal generator and discriminator, but there are several reasons why training a GAN like this is challenging and in most cases unstable. One of the more important reasons why this is the case is the problem of vanishing gradients. This was explained by Arjovsky et. al. [13]. They state, that data samples reside on a manifold in a high dimensional space $\mathbb{X}$ of all possible representations of a given data type, e.g. a $N \times N$ colour image. Typically this manifold only covers a small part of the entire space $\mathbb{X}$. Therefore the data samples and also the synthetic samples from the generator should only occupy a small part of $\mathbb{X}$. They then prove, that data in support $p_g(x)$ and $p_{data}(x)$ lie in a lower dimensional space than $\mathbb{X}$. This allows for both distributions to have zero overlap. In this case there would be a discriminator, that could tell apart real and synthetic samples with 100% accuracy. This would lead to the discriminator error converging to zero very quickly, which in return results in vanishing gradients used to update the generator-parameters and therefore no progress in generator training would be possible.
Another problem in GAN training with the above defined value function is, that train-

ing the Generator with a non-optimal Discriminator is less meaningful or even inaccurate. This can be resolved by exploring alternative cost functions.

2.4 Wasserstein-Generative-Adversarial-Networks (WGANs)

To solve the problems stated in the last section Arjovsky et. al. proposed the so called Wasserstein-GAN [14], which uses an approximation of the Earth Mover (EM) distance also called Wasserstein-1 distance

$$W(P_r, P_g) = \inf_{\gamma \in \prod(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\|\right] \tag{5}$$

where $\prod(P_r, P_g)$ stands for the set of all joint distributions $\gamma(x, y)$ whose marginals are the real probability distribution P_r and the generated probability distribution P_g. $\gamma(x, y)$ can be thought of as the mass needed to be moved to transform P_r into P_g and the EM-distance calculates the cost of the optimal transportation path. Since the infimum in equation 5 is hard to deal with, Arjovsky et. al. get rid of the infimum by using the Kantorovich-Rubinstein-duality [15], which results in

$$W(P_r, P_g) = \sup_{\|f\|_{Lip}\leq 1} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_g}[f(x)]. \tag{6}$$

This procedure approximates the EM-distance because it imposes a limitation on f to be 1-Lipschitz-continuous. The requirement can be expanded to include k-Lipschitz-continuous functions with a constant k, because this only adds a multiplicative factor k to Equation 6. This k-Lipschitz-constraint is best imposed upon the discriminator by adding a term to the loss function, which penalizes gradients unequal to 1. This gradient penalty loss function looks as follows:

$$GP(x_r, x_g) = \mathbb{E}[f(x_r)] - \mathbb{E}_{[}f(x_g)] - \lambda \cdot \mathbb{E}[(\|\boldsymbol{\nabla} f(\hat{u})\|_2 - 1)^2] \tag{7}$$

with λ being a hyperparameter to scale the gradient penalty. The mixture term

$$\hat{u} = \epsilon x_r - (1 - \epsilon) x_g \tag{8}$$

takes a random weighted average of the real data samples x_r and the generated samples x_g. The weight ϵ is sampled from a uniform distribution with values between 0 and 1.

Using the approximate Wasserstein-distance as the loss function in GAN training solves many problems from the orginal GAN setup. Firstly, it is more suited for GAN training because it converges more consistently than other loss functions, which is shown in [14]. Secondly, it solves the problem of vanishing gradients because of the k-Lipschitz-

constraint.

2.5 Applications of GANs in a physics context

As already mentioned in Section 1.1, since the first paper introducing the GAN structure by Goodfellow et. al. [12] was published in 2014, GANs have been used in multiple different applications with a physics context and have shown very promising results. For example the GAN framework was first used in a HEP context by Paganini et. al. in 2018 [16]. They tried to simulate 3d calorimeter particle showers for LHC detectors using the standard GAN architecture. They achieved a speedup of $\mathcal{O}(10^2)$ compared to the classical simulation, but the GAN could not reproduce the simulation with the necessary accuracy. This process was later improved by Erdmann et. al. [17]. They used the WGAN architecture and improved the speedup factor, which they state to be between 10 and 6660 compared to the simulation, depending on the simulated shower energy and the computer setup used to run the WGAN. Their results also show, that the trained WGAN produces samples, which show good agreement with the classical simulation in various typical observables for calorimeters.

3 Applying GANs to noise from radio detectors

This section focuses on constructing and training a WGAN for radio detector noise. It also includes a discussion of the structure of the data set used for training and a comparison of the training performance of the different networks.

3.1 The training data set

As already mentioned in section 1.3, the data set used to train the GAN was recorded by one of the stations at the ARIANNA detector. The data set used is run 225 from station 32, which contains events recorded between 22.02.2017 and 04.04.2017. The station consists of four LPDAs, which are recording events in 256 time bins at a sampling rate of 1GHz. The data set contains 109446 events with four noise traces in each event. For the detector to record an event, the recording needs to be triggered. There are two main types of triggers used by ARIANNA. The first one is the forced trigger, where the detector is read out periodically without a significant increase in signal intensity in the detector. The other trigger type is called thermal trigger. Here the detector is read out, when an antenna measures an intensity, that surpasses a threshold. Out of the 109445 events in run 225, 46336 of the events were triggered by a forced trigger and 63109 events by a thermal trigger. This section will cover important characteristics of the data set, which should be reproduced by a well performing GAN.

Figure 6 shows four noise traces randomly sampled from different events and a histogram, which shows the amplitude distribution of all individual time bins in the entire data set, as well as only forced triggers and only thermal triggers. One can see that the intensity varies strongly between different traces. The distribution shows a rather complex shape with a primary peak at 25 mV, which indicates, that there is a DC-component in the data, and two side peaks symmetrically placed at approximately ± 550 mV. One can see, that these side peaks only show up in thermal trigger events.

Another important characteristic of this type of data is its frequency spectrum. The transformation between time domain and frequency domain is performed via Fourier transformation with the FFT algorithm. To achieve power conservation in the Fourier transform the spectra are normalized by adding a multiplicative factor of $\sqrt{2}$ when transforming from time to frequency space and a factor of $\frac{1}{\sqrt{2}}$ for the inverse transformation. The average spectrum of the data set is depicted in Figure 7. It shows a large peak at 0 MHz, which stems from the DC-component. More importantly the spectrum has a large 3-peaked structure at 100 MHz and another smaller peak at 190 MHz. Beyond 200 MHz the frequency intensity falls off to an almost constant value with small variations.

In Figure 8 the frequency spectra of 1000 different traces are plotted next to each other in a color plot. Yellow color represents high intensity and blue color low intensity. The horizontal yellow lines show the frequencies, which are present with high intensity in all traces. The vertical yellow lines visible in the plot suggest, that the overall intensity of traces over the entire frequency spectrum varies strongly across different events. This vertical structure is best visible around 100 MHz, because the antenna response is largest in this frequency band.
Another thing to look at is, where the event-to-event intensity variability comes from. 500 randomly selected forced triggered events and 500 randomly selected thermal triggered events are plotted next to each other in Figure 9 . The first 500 events are forced trigger events and the second 500 events thermal trigger events. One can see, that the forced trigger events show consistently low intensity compared to the thermal trigger events. The thermal trigger events show small variations in intensity, but the variability is not as strongly present as before sorting the events by their trigger. This means, that the intensity variability of the data set mostly stems from the differently triggered events in the data set. Therefore, when a generator network learns to correctly reproduce this variation in intensity, it also learns to reproduce the ratio between forced and thermal trigger events.
It is also interesting for training, whether there is a difference between the traces from different channels. In Figure 10 the amplitude distribution of all time bin values in the

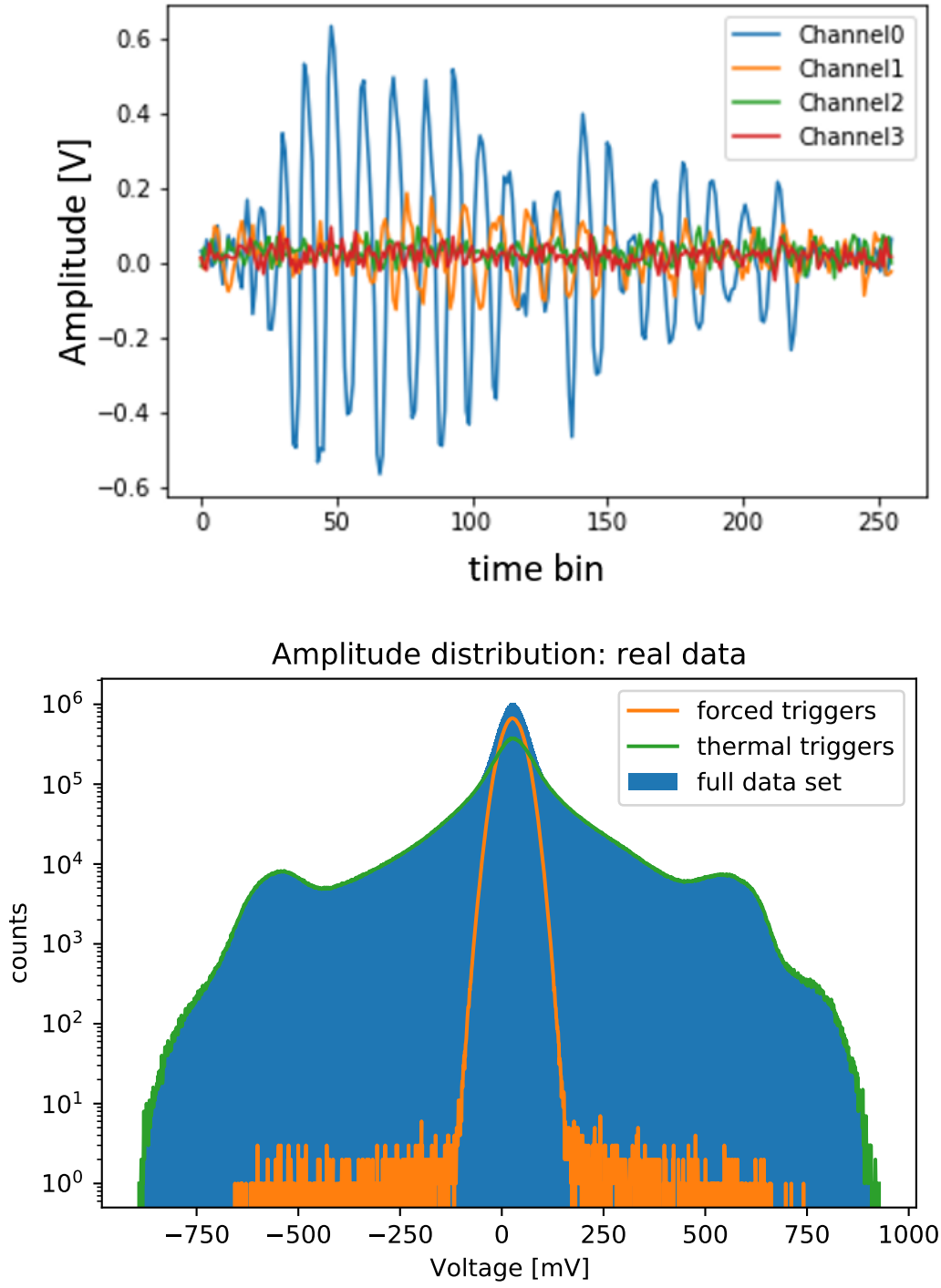

Figure 6: Top: Noise traces randomly sampled from different events; Bottom: Amplitude distribution of the full data set (blue), only forced triggers (orange) and only thermal triggers (green)

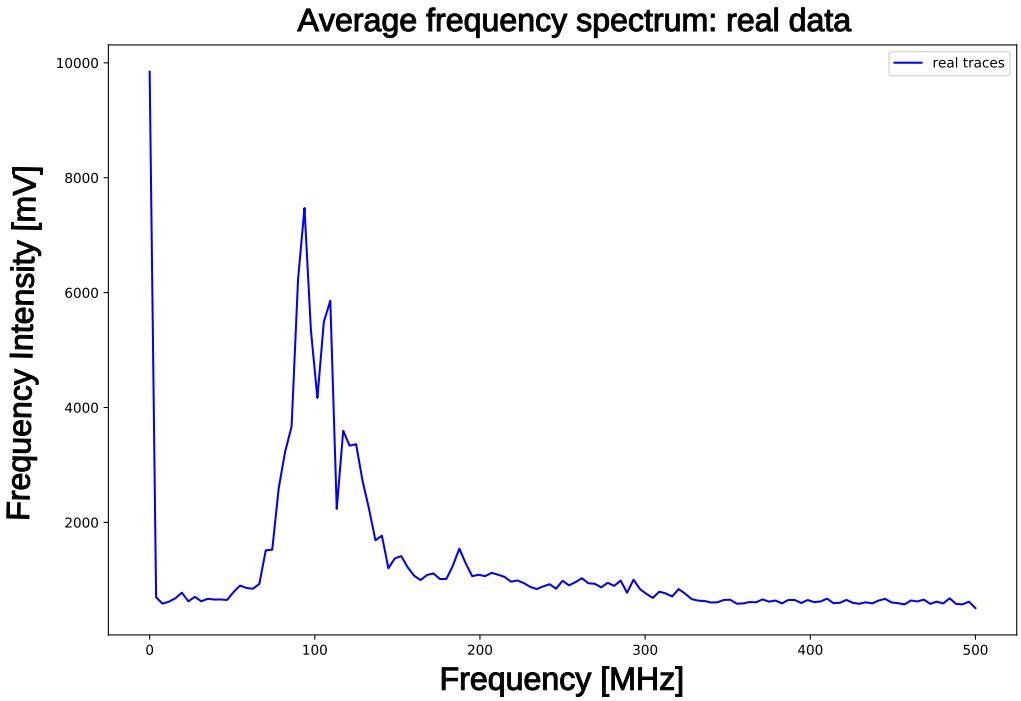

Figure 7: Average frequency spectrum of the data set

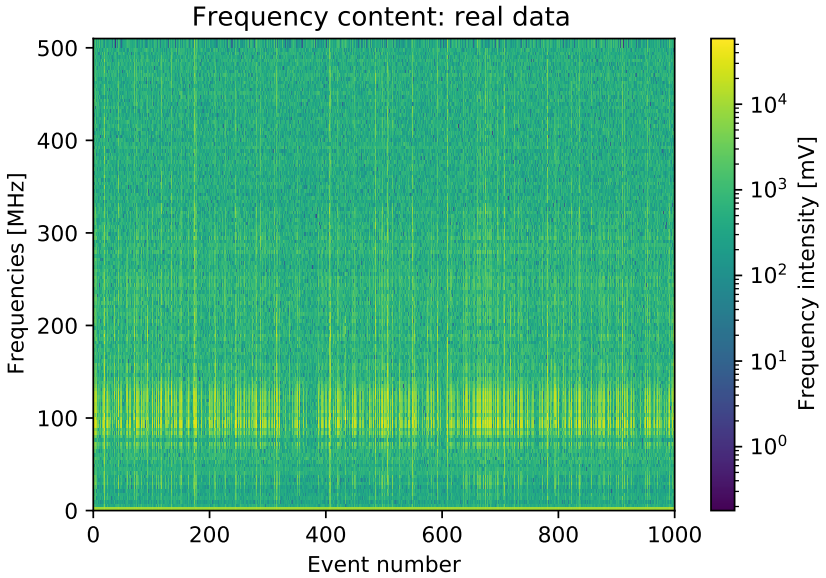

Figure 8: Frequency spectra from different traces

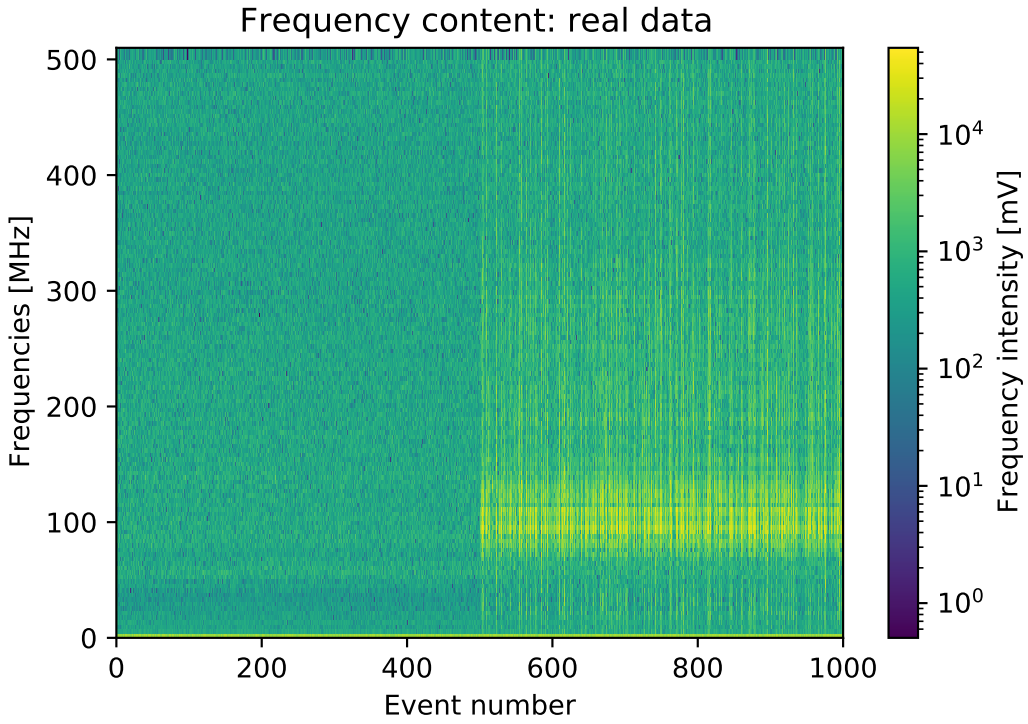

Figure 9: Frequency plot of 1000 different traces; the first 500 traces are forced trigger events, the last 500 are thermal trigger events

data set is depicted for each channel separately. One can see, that there is no significant difference between the four channels distributions. Figure 11 shows the average frequency spectra of the four channels separately. Looking at the different spectra one can see, that all spectra have similar shapes. Channel one, three and four also have similar overall intensity. Channel two shows a lower intensity at around 100 MHz than the other three channels.
Overall, one can see, that both the amplitude distribution of the time bins and the frequency spectra are similar for all channels. Therefore it is reasonable, to train the network on traces from all channels without differentiating between them.

3.2 Designing the Network Architecture and the Training Loop

The programming for this thesis is done with python and utilizes tensorflow's Keras API for its ML and NN algorithms. Building a WGAN consists of two main steps. Firstly, one has to choose the network structure for both the generator and the discriminator, which is also called critic, in the WGAN context. In this thesis two network architectures are compared. The network structure of the first model utilizes only fully connected layers, which are called dense layers in Keras. The networks used are depicted in Figure 12. This type of figure shows the layer type, the input shape and the output shape of each layer in a network. The arrows also indicate the order, in which

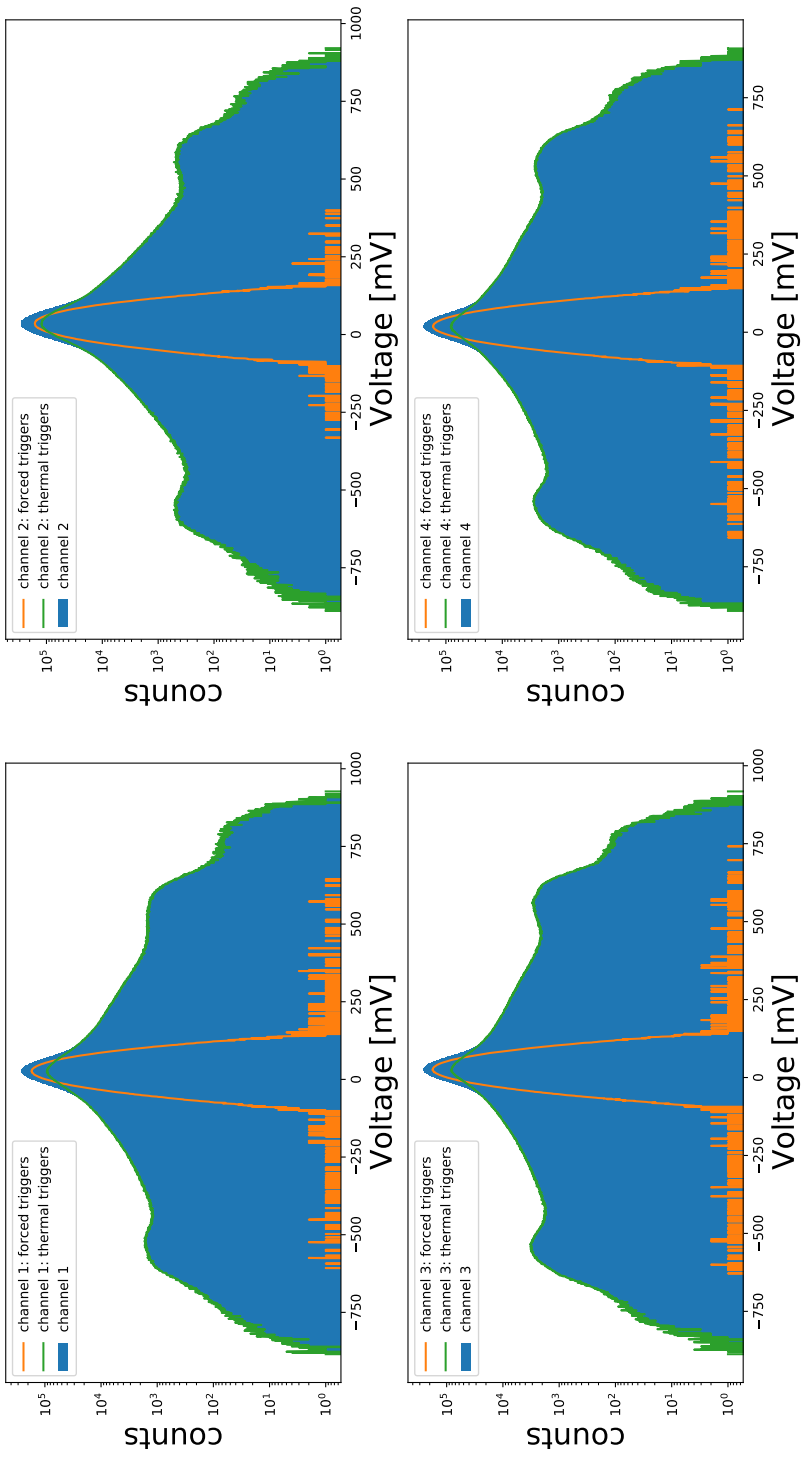

Figure 10: Amplitude distribution of the four channels separately; all triggers (blue), only forced triggers (orange), only thermal triggers (green); Top left: channel 1, Top right: channel 2, Bottom left: channel 3, Bottom right: channel 4

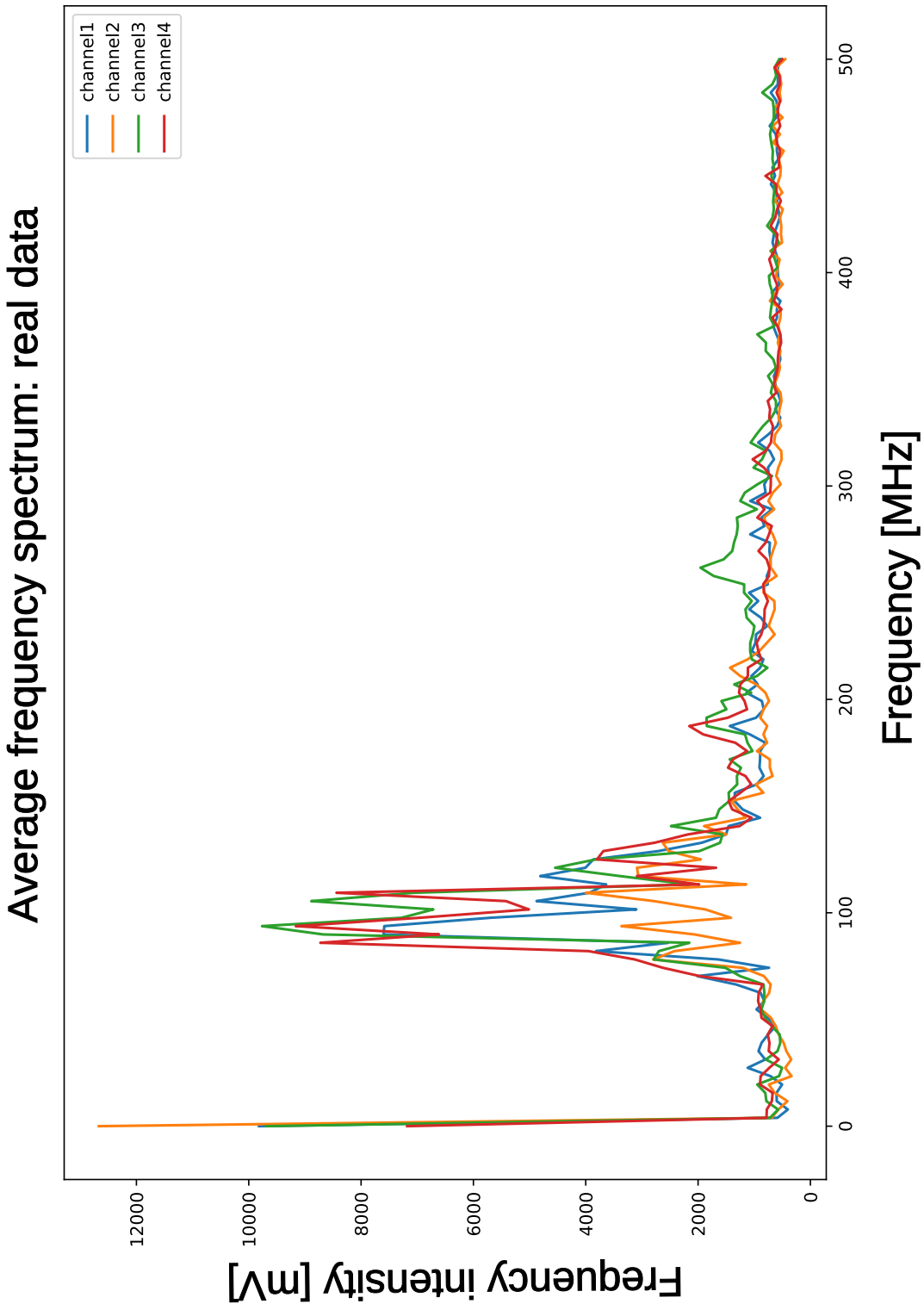

Figure 11: Average frequency spectrum of the four different channels; channel 1 (blue), channel 2 (orange), channel 3 (green), channel 4 (red)

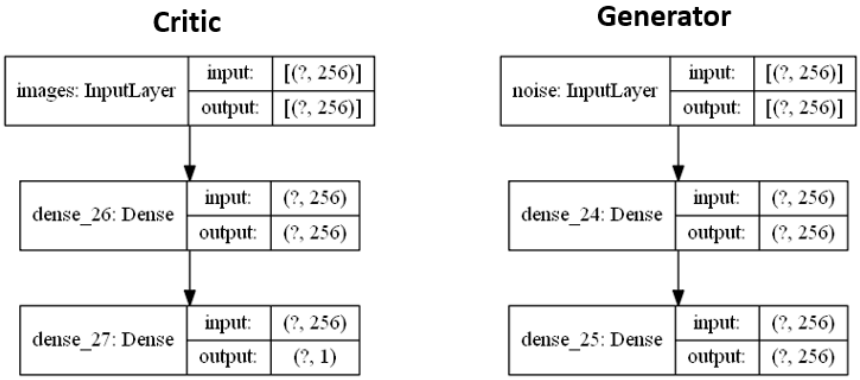

Figure 12: Fully connected network architecture; Left: Critic Network, Right: Generator Network

the input gets passed through the network. Both the generator and the critic take an input array of shape (batch-size,256), where the batch-size represents the number of outputs the network generates. For example, if batch-size=1, the generator produces one synthetic trace, if batch-size=1000, 1000 traces are generated. Both networks also use one hidden dense layer. The output layer of the generator produces an array with shape (batch-size, 256), since the actual traces contain 256 time bins each. The critic output is an array of shape (batch-size,1).

The second model adds a convolutional layer to the prior model. This model is shown in Figure 13. This layer applies filter matrices to the input, which aids in pattern recognition and allows the network to learn local structures more easily than a network utilizing only fully connected layers. An average pooling layer was also added to the generator to reduce the dimension and thereby the amount of network parameters, which improves computation speed. The pooling layer takes the average value of three adjacent neurons.

The second step is to write a training loop for the WGAN. This includes choosing the (hyper-)parameters, defining the training models for critic and generator and then performing the training. For this thesis the 'Wasserstein GAN for physics simulation' example from RWTH Aachens VISPA cluster [18] was used as a starting point for the training loop. The training loop in the VISPA example was constructed to reproduce 2D images instead of the 1D traces of the radio antennas covered in this thesis. Therefore some of the functions an plotting routines needed to be modified in order to accommodate the one-dimensional input instead of the two-dimensional data. The structure of the training algorithm is described in Figure 14. The hyperparameters were not changed from those stated in Figure 14. The batch size m was chosen to be either 100 or 200 and training was done for 10 epochs, which means that the entire

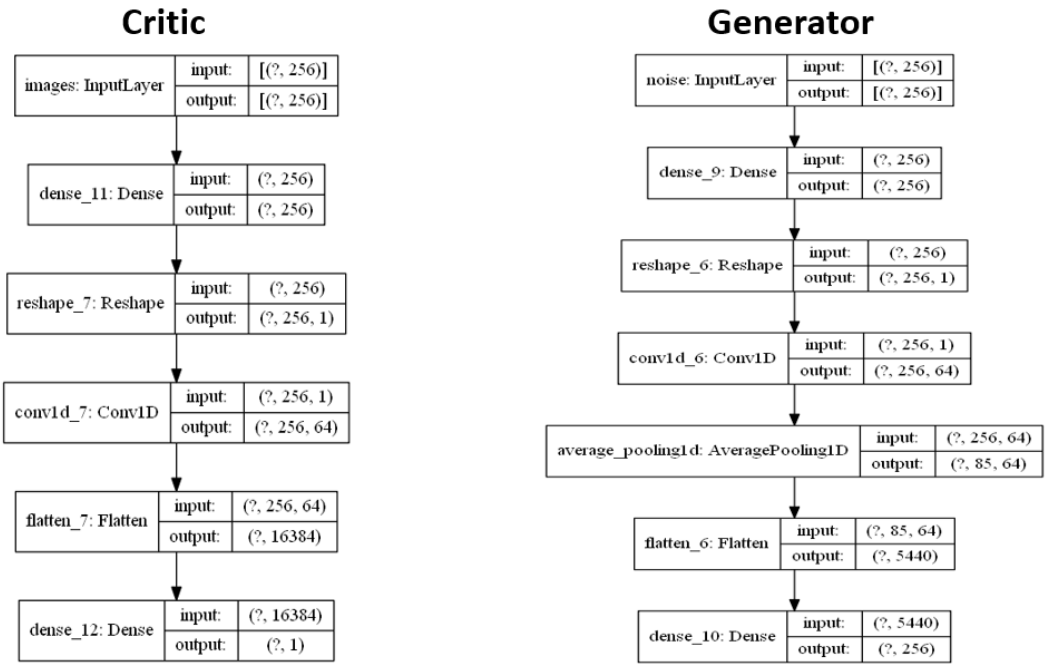

Figure 13: Convolutional model; Left: Critic, Right: Generator

training data set was passed through the algorithm 10 times. The training is performed by first training the critic n_{critic} times by taking m real data samples, generating m samples with the generator and sampling the random weight ϵ for the gradient penalty. Then the gradient penalty loss from Equation 7 is calculated and the critic parameters w are adjusted with this loss function by the Adam optimization algorithm. After the critic training the generator gets trained once by generating a batch of synthetic samples, which is then passed through the critic. The critic output is then used as the loss function and the parameters of the generator are adjusted using the Adam optimizer.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 14: Description of the training algorithm [19]

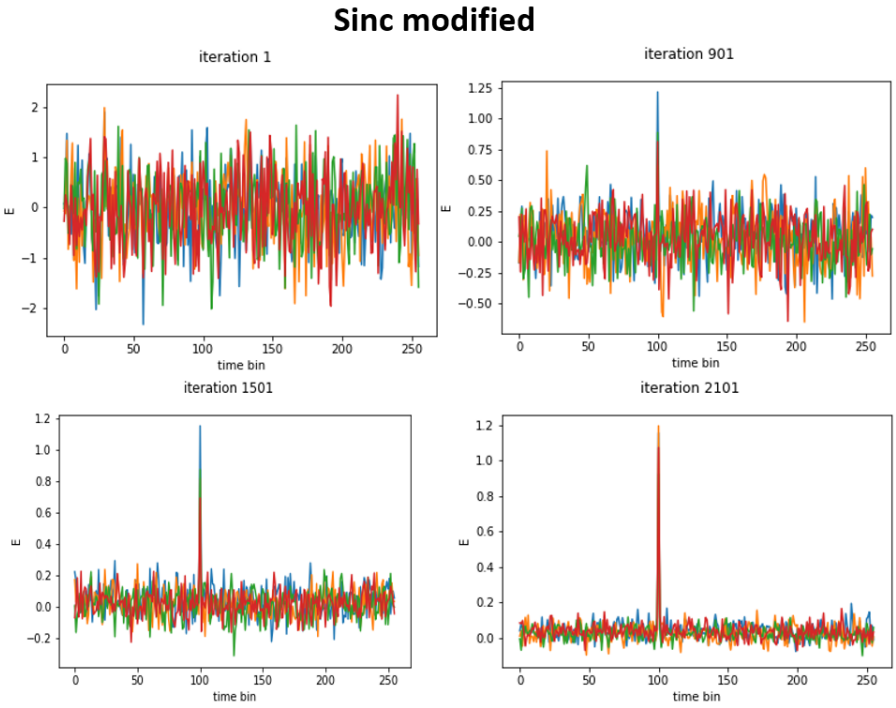

Figure 15: Training progress with added sinc-pulse

3.3 Training

To check whether the simple fully connected model is capable of learning the structure of the data passed to the training loop, a sinc-pulse was added to the training data beforehand. The peak of the pulse was placed at time bin 100 and added 1 V to the measured intensity of that bin. This added a structure to the data, which can be easily identified. Figure 15 shows how the network progressively learns to place a sinc-pulse with the right intensity at the correct bin.

After that the training of the dense-layer model was performed with unmodified data. For this training process the batch size was chosen to be $m = 100$. The training data was fed into the training loop without randomizing the sequence of events. Figure 16 shows the loss curves for both the critic and the generator. The plot of the critic loss shows the total loss as described in Equation 7 in red and also the Wasserstein-loss and the gradient penalty term separately. The critic loss shows a periodically recurring spike and the generator loss is converging to a value close to zero. This is problematic because the critic should theoretically converge to an optimal state with a loss close to zero and the generator loss should not converge.
As already mentioned in section 3.1 the data set used for training covers a time span of approximately 6 weeks. Therefore it is likely, that the data set, in its original order, contains some sort of time dependent variability. It is thereby reasonable to assume that the periodic structure in the critic loss in Figure 16 exists due to this variability.

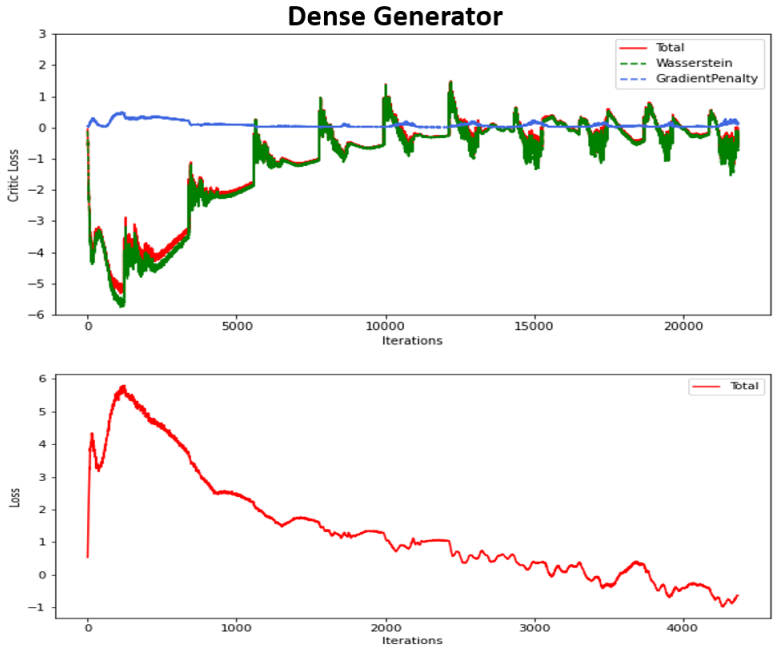

Figure 16: loss curves of the dense network (top: critic, bottom: generator)

To test this hypothesis, the training method was modified by randomizing the sequence of the data set. By doing this any time dependencies in the training data should disappear and result in a converging network. As seen in figure 17 performing the training this way allowed the critic to converge and removed the periodic structure in the loss curve. It also improved the training performance of the generator.

The next step was to train the CNN. The sequence of the data set was also randomized, because it yielded good results with the fully connected network. The batch size was set to $m = 200$, because training of NNs is a statistical process and therefore having a bigger batch should result in a more stable training process. Another important thing to note about the training process is, that the data was read in with a different reader than the one used for the other networks. This alternate noise reader allows to read in the data set in smaller batches, which results in only the data needed for 1 training iteration being stored in memory at once. This was necessary because the CNN produces large arrays in the convolutional layer, which require more memory and having the entire data set read in and stored in memory in addition to the memory demands of the network would have exceeded the capacity of the computer used for training. This new reader also creates new problems. Firstly, not having the entire data set read in for the entire training means, that instead of reading in the data once in the beginning, the entire data set has to be read in for each epoch of training, which slows down the training loop. Secondly, the bunches are read in in the original order of the data set. For example if a data set would be divided into ten bunches, the data

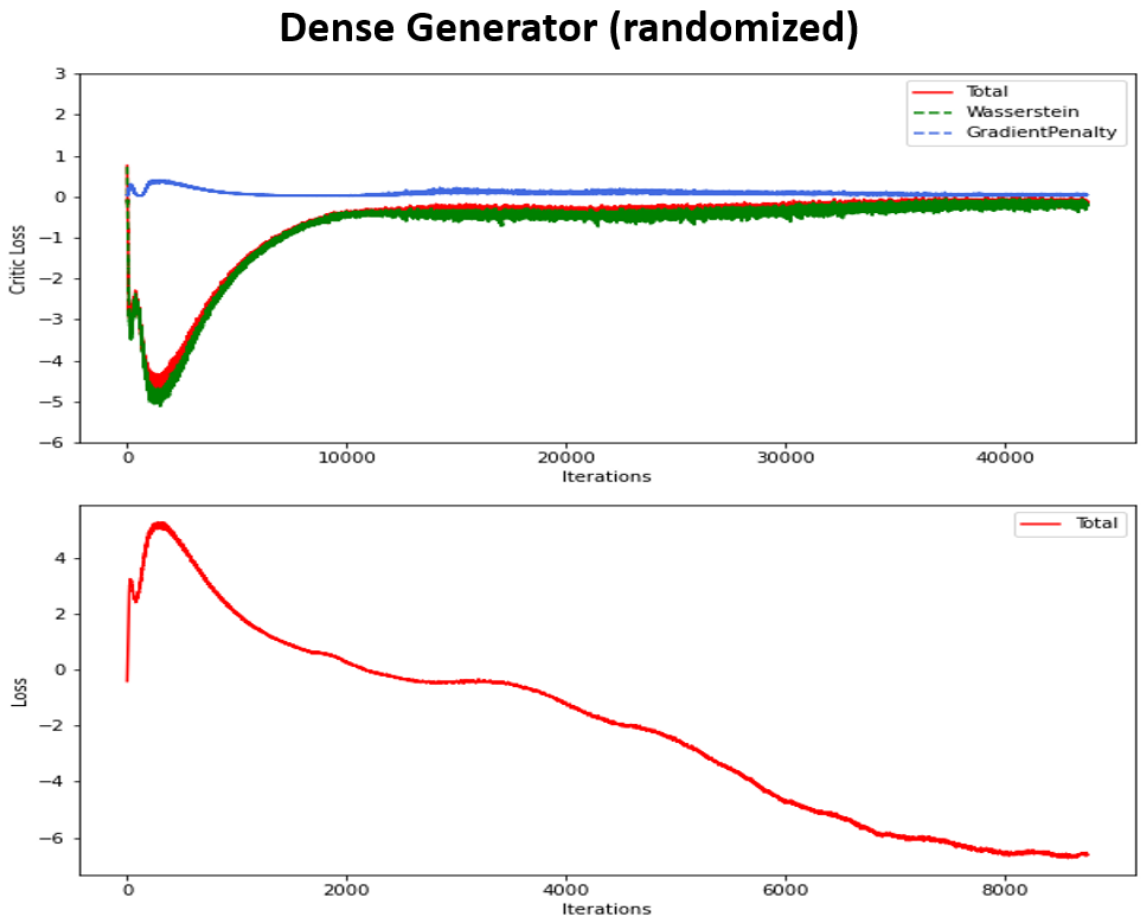

Figure 17: loss curves from training with a randomized data set

in the bunches would be in the original order and the reader only randomizes the order in which the bunches are returned, e.g. in one iteration of the reader the first bunch to be returned could be the sixth bunch and in another iteration it is the fourth bunch. It is then possible to randomize the order in each bunch separately but fully randomizing the order is not possible.

Figure 18 shows the loss curves for the CNN-GAN. One can see, that there is a strong variability in the Wasserstein- and the GardientPenalty-term of the critic loss. This sort of behavior is typical for more complex network structures, where its more likely for the optimizer to find a parameter configuration, which is good at minimizing the Wasserstein loss but does not fulfill the k-Lipschitz-criterium and is therefore penalized by the gradient penalty. This is why there are these spikes in the critic loss curve. The minimization of the generator loss was quite successful, but this has to be taken with a grain of salt because the critic had not converged after the training loop and therefore the generator loss might not be calculated based on an optimal critic. Further training could be necessary to fully optimize the network.

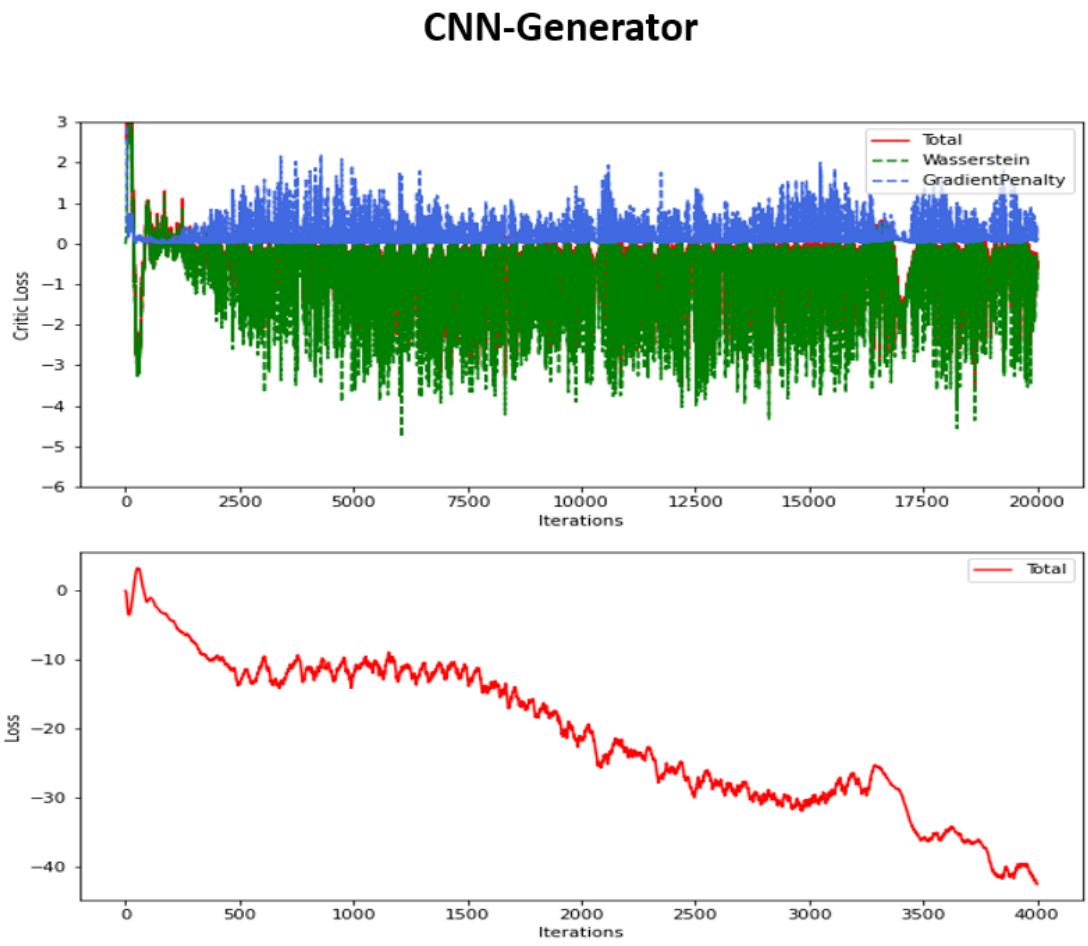

Figure 18: Top: critic loss curve of the CNN; bottom: generator loss curve of the CNN

During the analysis of the network performance concerning physical properties of the generated samples, which will be covered in the next section, it was discovered, that the networks struggled to fully capture the structure of the actual data in the frequency domain. To try and help the network in learning this structure it was tested, whether modifying the data in the frequency domain could improve training performance. In section 3.1 it was already mentioned, that the data contains a DC-component. This DC-offset contains no important information for reconstructing the neutrino events. Therefore removing this DC-offset from the data could allow the network to focus on learning the important characteristics of the data set. To test this hypothesis the data was modified in the frequency domain before training. A Fourier transformation was performed on the data, the bin representing the DC-offset was set to 0 and then the spectrum was transformed back into time domain and used for training. Due to the high computational demand of training the CNN, this process was only tested on the fully connected network. Figure 19 shows the loss curves from training the dense network with the Fourier modified data. One can see, that the overall shape of the critic loss looks similar to the shape of the dense generator trained on data with the DC-offset. The critic loss shows variability similar to the loss curve of the CNN critic

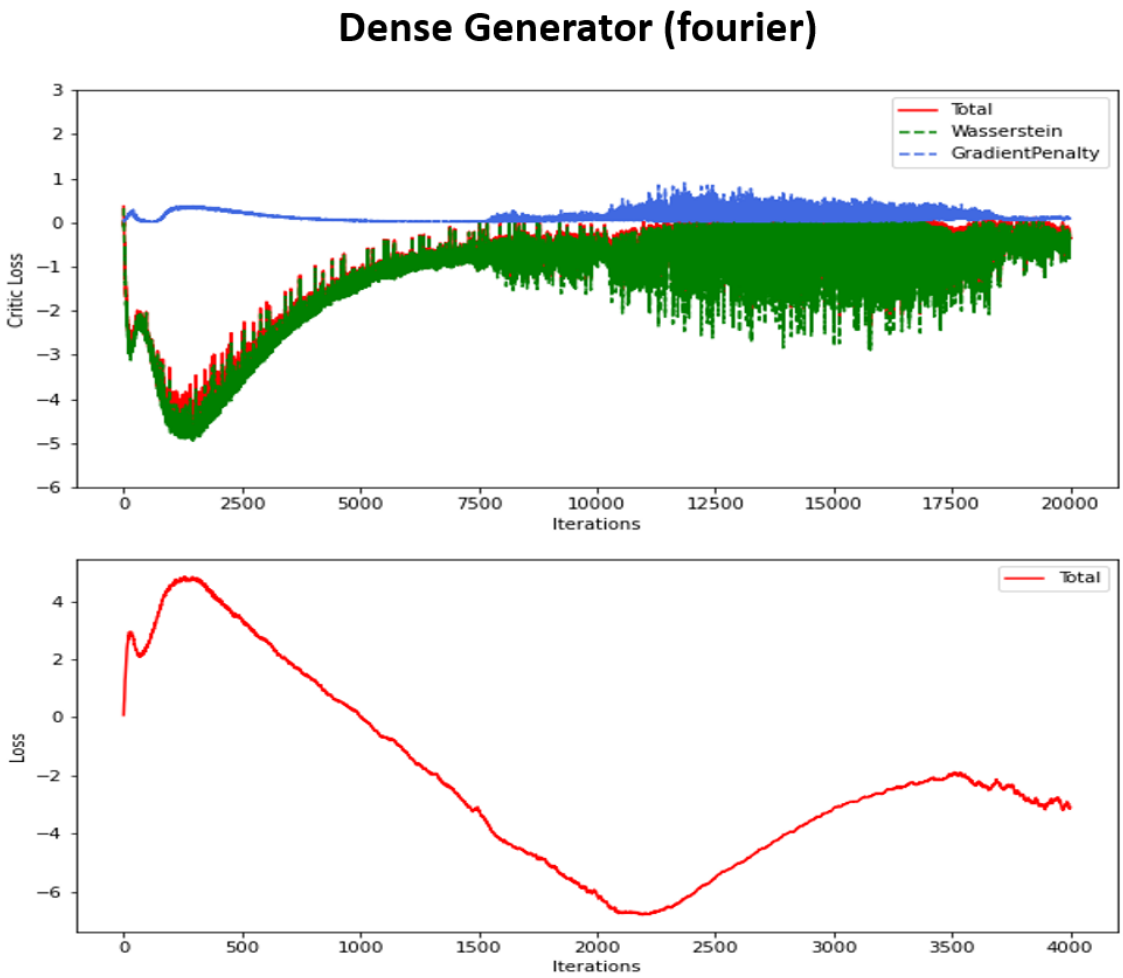

Figure 19: Top: critc loss curve for the Fourier modified training; bottom: generator loss curve for the Fourier modified training

but opposite to the CNN critic the variability of the critic loss decreased again in the last training iterations for the dense network. The shape of the generator loss follows the development of the critic loss. At around iteration 2100 the generator loss has a minimum. This position corresponds to approximately critic iteration 10500, because for each generator iteration the critic is trained five times. After this point the critic goes into a phase of low loss, which corresponds to a strong critic and therefore the generator loss rises. Once the critic loss approaches zero again the generator loss starts going down.

4 Performance of the GANs

The following section will cover an analysis of the performance of the different networks and training methods. First, the accuracy of the generators will be analysed by looking at important physical characteristics of the real data set and how well the synthetic traces reproduces them. After that it will be discussed, whether and if so how much

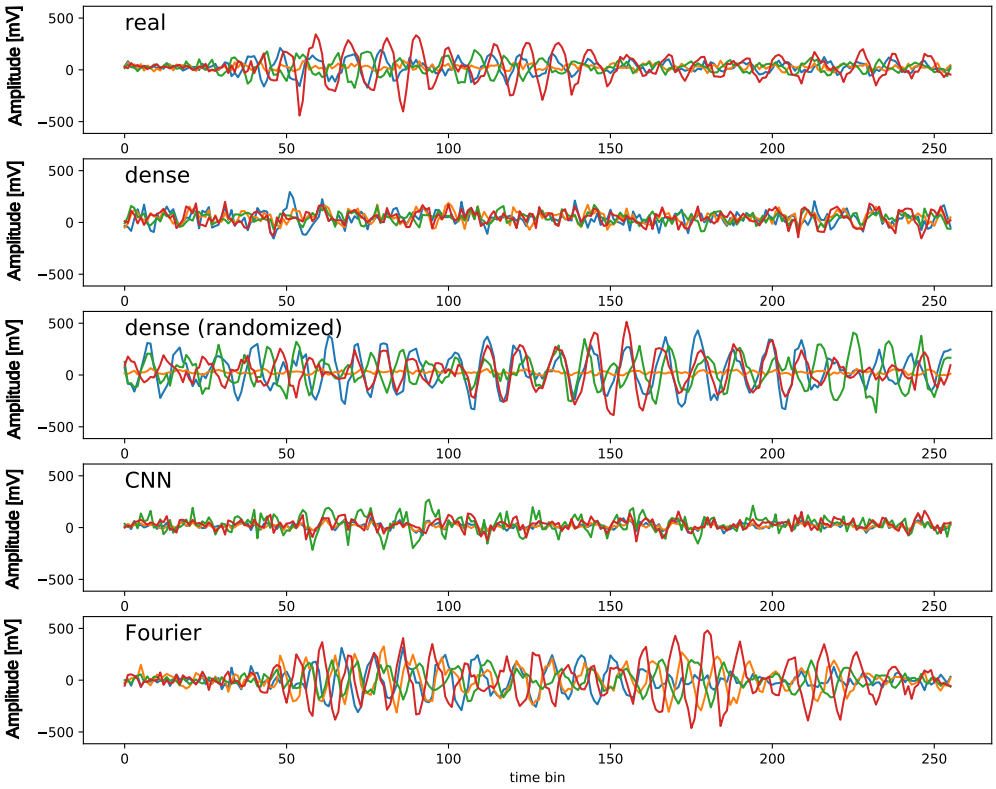

Figure 20: Noise traces sampled from real data and the different NNs. Order from top to bottom: real data, dense GAN, dense GAN trained on the randomized data set, CNN GAN, dense GAN trained on the Fourier modified data set

the computational performance of the GANs improve on the currently used method to get realistic noise.

4.1 By eye comparison

First, real and synthetic noise traces are inspected visually. Figure 20 shows noise traces sampled from the real data and the generators. One can see, that real traces vary in intensity. Some traces are low intensity and have maximum amplitudes close to zero, other traces amplitudes get as large as 400-500 mV. Looking at traces produced by the generators one can see, that the dense generator trained without the randomized data set and the CNN generator do not produce enough traces with high intensity. The dense generator trained with randomized data and the one trained with Fourier modified data produce traces of the correct intensity and intensity variability. By eye they could be mistaken for real noise traces.

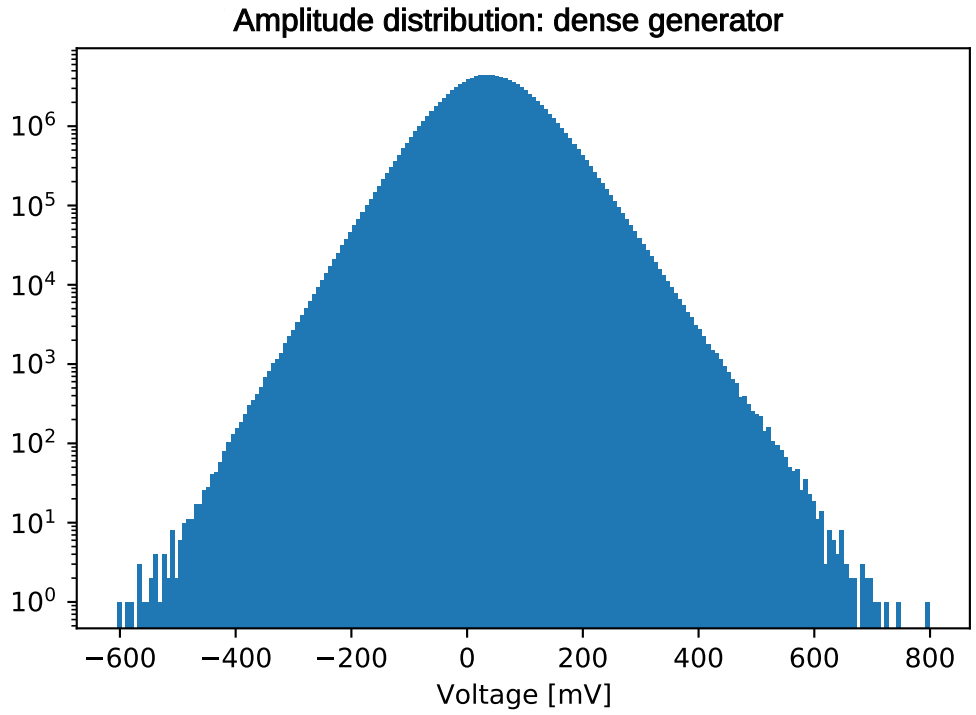

Figure 21: Amplitude distribution of the dense generator

4.2 Amplitude distribution

The next step in analysing the accuracy of the synthetic data is to look at the overall amplitude distribution in the data set. As already seen in Figure 6, the histogram of all time bin values of the data set has a characteristic 3-peaked shape with peaks at 25 mV and ±550mV. First, the amplitude distribution of the dense generator, which is depicted in Figure 21, will be compared with the actual data. Looking at the histogram it becomes apparent, that this generator is not able to reproduce the real distribution accurately. Instead of the three expected peaks, there is only one at 35 mV. The shape of the center peak is also not as sharp as the one of the real distribution.

The voltage distribution of the dense GAN trained on randomized data is shown in Figure 22. This distribution looks closer to the actual data than the one from the GAN trained on non-randomized data. The generator reproduces the sharp peak at 23 mV. Other than that there are still some differences in the distributions. The generator was not able to capture and reproduce the three-peaked shape of the distribution. Also the minimal value present in the generated distribution is $U = -1500$ mV, which is almost two times the minimal value occurring in the real distribution.

The distribution synthesized by the CNN generator, shown in Figure 23, looks similar to the one generated by the dense generator, which was trained on non-randomized

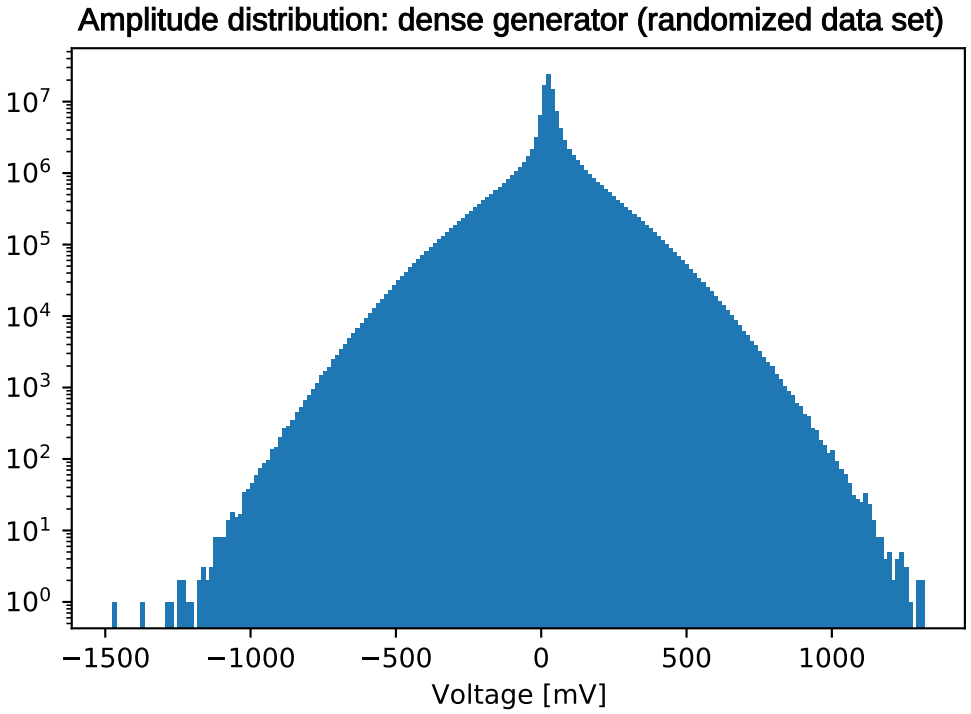

Figure 22: Amplitude distribution of the dense generator trained on the randomized data set

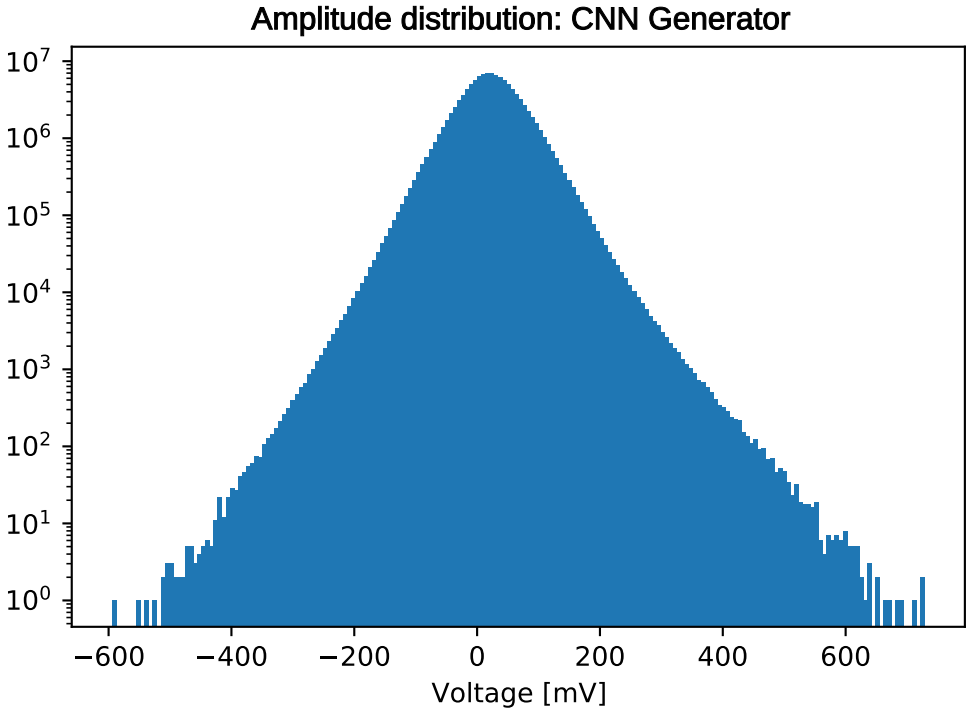

Figure 23: Amplitude distribution of the CNN generator

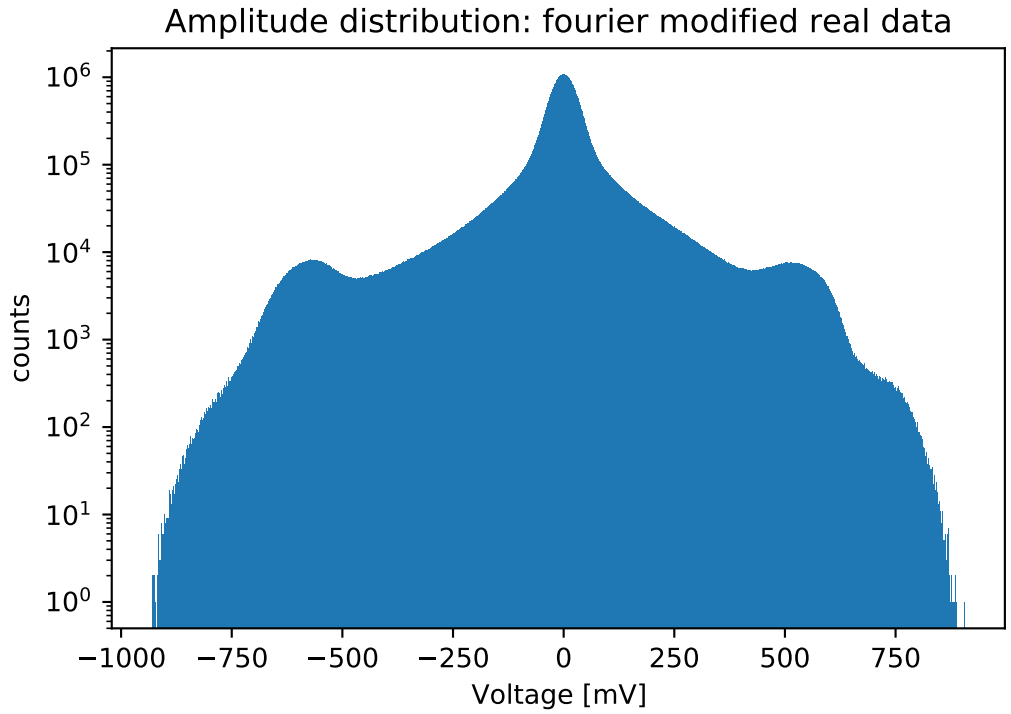

Figure 24: Amplitude distribution of all time bins in the Fourier modified data set

data. The center peak lies at 19 mV. The deviations of the generated distribution from the real data are therefore also the same as for the dense non-randomized generator.

Next, the performance of the dense generator trained on Fourier modified data is analysed. To reasonably compare the synthetic traces to real data, one has to look at the Fourier modified data. In Figure 24 one can see, that the shape of the amplitude distribution is not affected by removing the DC-offset from the data. Only the position of the three peaks shifted. The central peak lies at -0.15 mV, the two side peaks at ~ 520 mV and ~ -580 mV. Figure 25 shows the distribution of the dense GAN trained on the Fourier modified data set. Other than a sharp center peak there is no similarity between this distribution and the actual data distribution. The generator places the central peak accurately at around -1 mV.

Overall, the dense generator trained on the randomized data set captured the real distribution best compared to the other generators. But even the best result was not able to reproduce the real distribution accurately. This problem may be resolved by adding more complexity to the networks, for example by adding more layers.

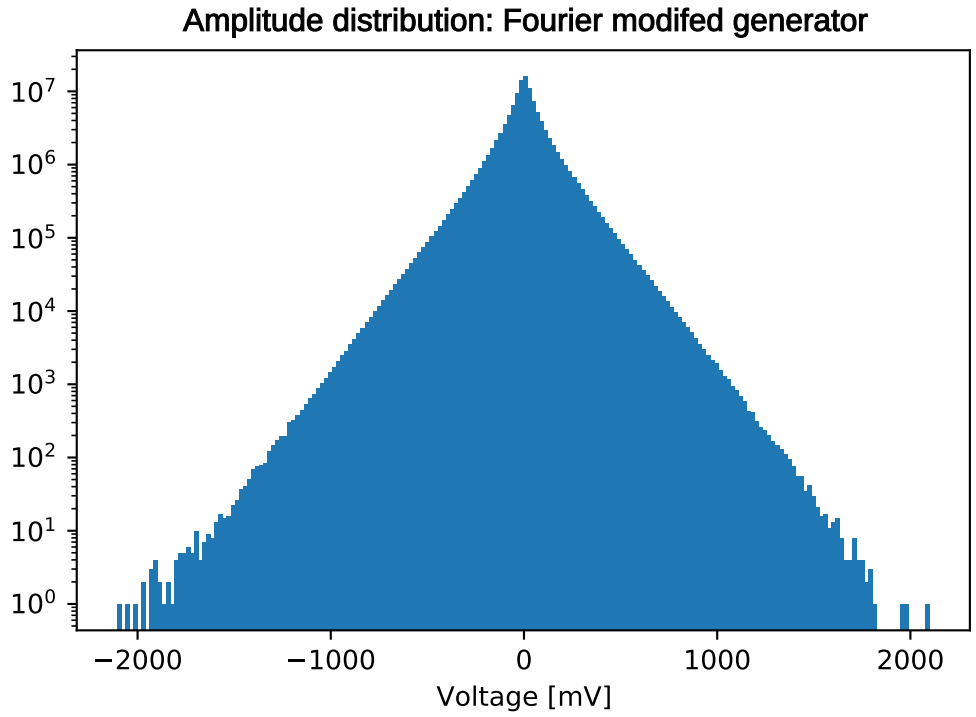

Figure 25: Amplitude distribution of the dense generator trained on the Fourier modified data set

4.3 Frequency spectrum

Another important characteristic of the real data is its frequency spectrum. To check if the generated traces match the real noise traces in frequency space the average frequency spectrum of synthetic traces is compared to the average spectrum of the real data. To do so, a sample of 10000 synthetic spectra and real spectra were averaged and then compared.

Beginning with the non-randomized dense generator, whose spectrum is depicted in Figure 26, one can see, that compared to the spectrum of the real traces the synthetic spectrum has a more intense DC-Offset and higher intensity for frequencies beyond 200 MHz. The shape of the most prominent part of the spectrum around 100 MHz is reproduced by the generator almost perfectly, but its intensity is lower than in the real spectrum.

Figure 27 shows a frequency color plot comparable to the one in Figure 8, which shows 1000 spectra of real noise traces. One can see, that the intensity of single traces sampled from the dense generator does not vary as strongly as is does for real traces. Especially when looking at the intensity at 100 MHz the intensity is almost constant in the synthetic sample, whereas in the real data this part of the spectrum the intensity varies strongly.

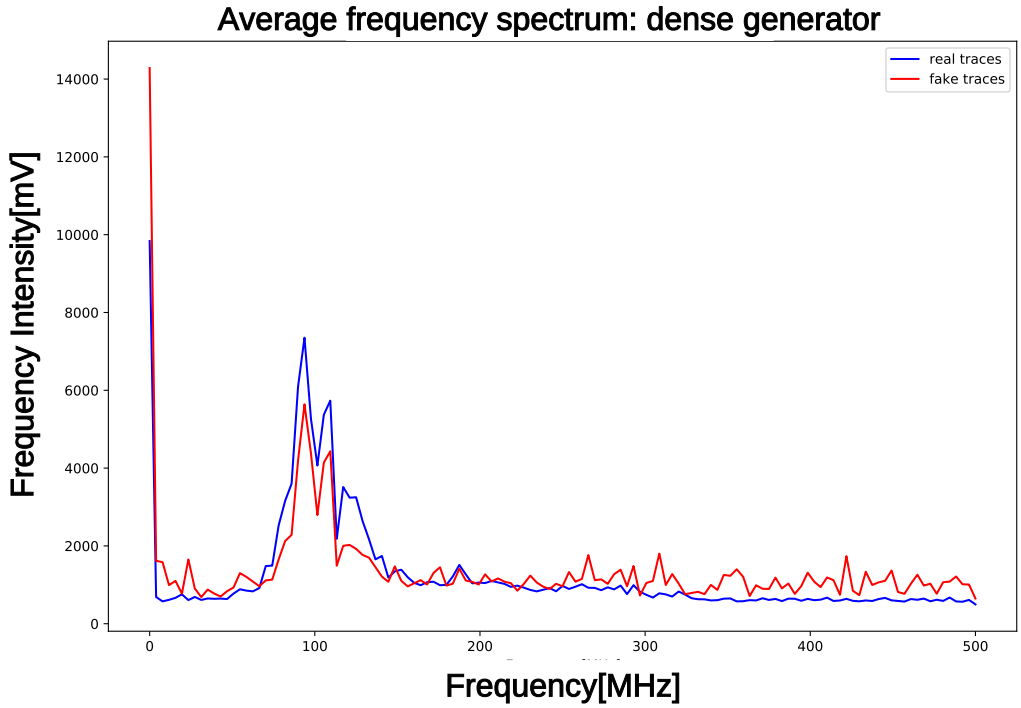

Figure 26: Average frequency spectrum of the non-randomized dense generator (blue: real traces; red: synthetic traces)

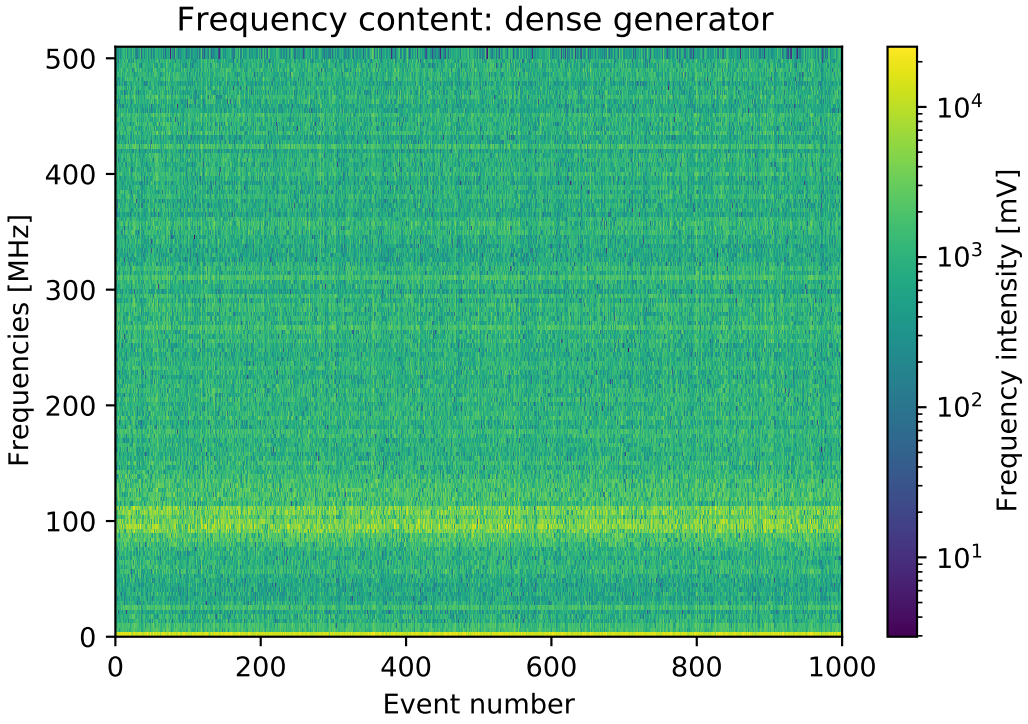

Figure 27: Frequency spectra from different synthesized traces from the dense generator

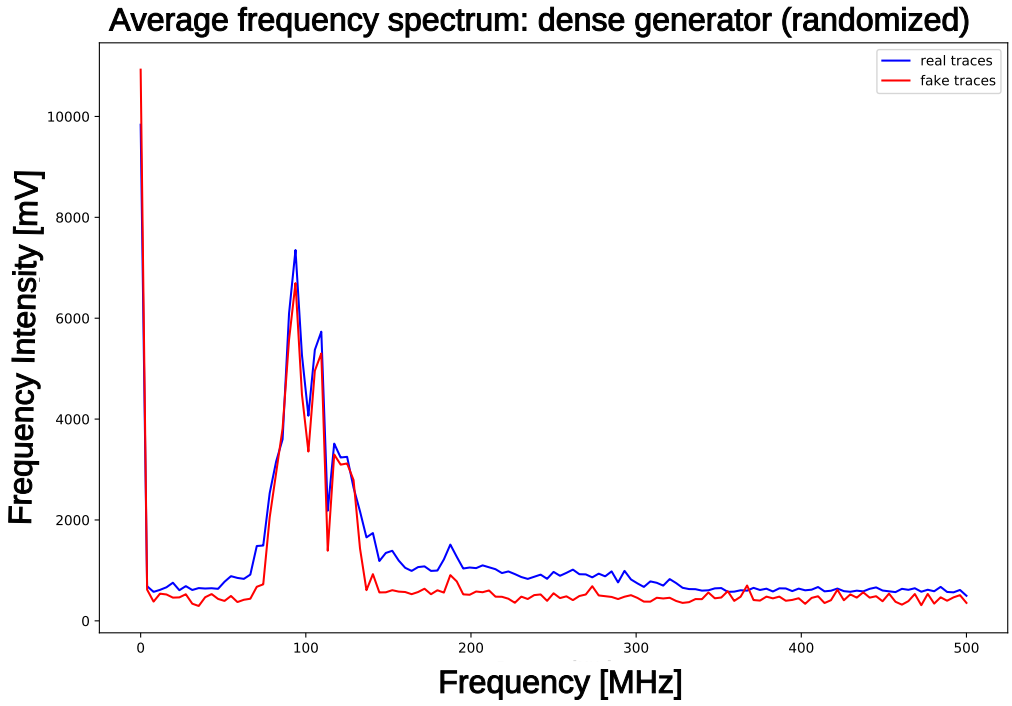

Figure 28: Average frequency spectrum of the randomized dense generator (blue: real traces; red: synthetic traces)

Next, we take a look at the spectrum of the dense generator trained on the randomized data set shown in Figure 28. The DC-component of the synthetic spectrum is still a bit more intense than in the real data but not as extreme as without the randomization. The shape of the generated spectrum at 100 MHz matches the real data almost perfectly both in shape and intensity. The rest of the synthetic spectrum is also close to the shape of the real data, but the intensity is lower.

As can be seen in Figure 29, the randomized dense generator also produces traces with varying intensity as it would be expected from actual data.

The average spectrum of the CNN generator is depicted in Figure 30. The synthetic spectrum does not match the real spectrum. The intensity and the shape of the structure at 100 MHz are not similar to the real shape. There is also an intensity spike at approximately 490 MHz, which does not appear in the real spectrum.

Figure 31 confirms what was already visible in the average spectrum. The synthesized traces from the CNN contain more frequencies beyond 150 MHz. The plot also shows, that the generated traces do not vary in intensity as much as the real traces.

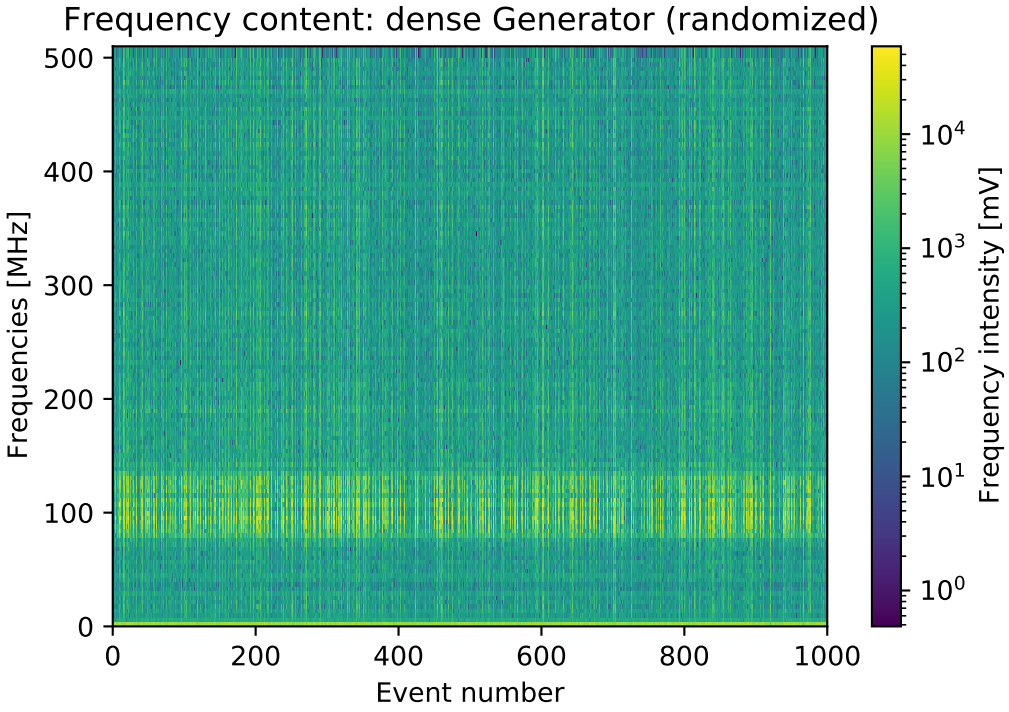

Figure 29: Frequency spectra from different synthesized traces from the randomized dense generator

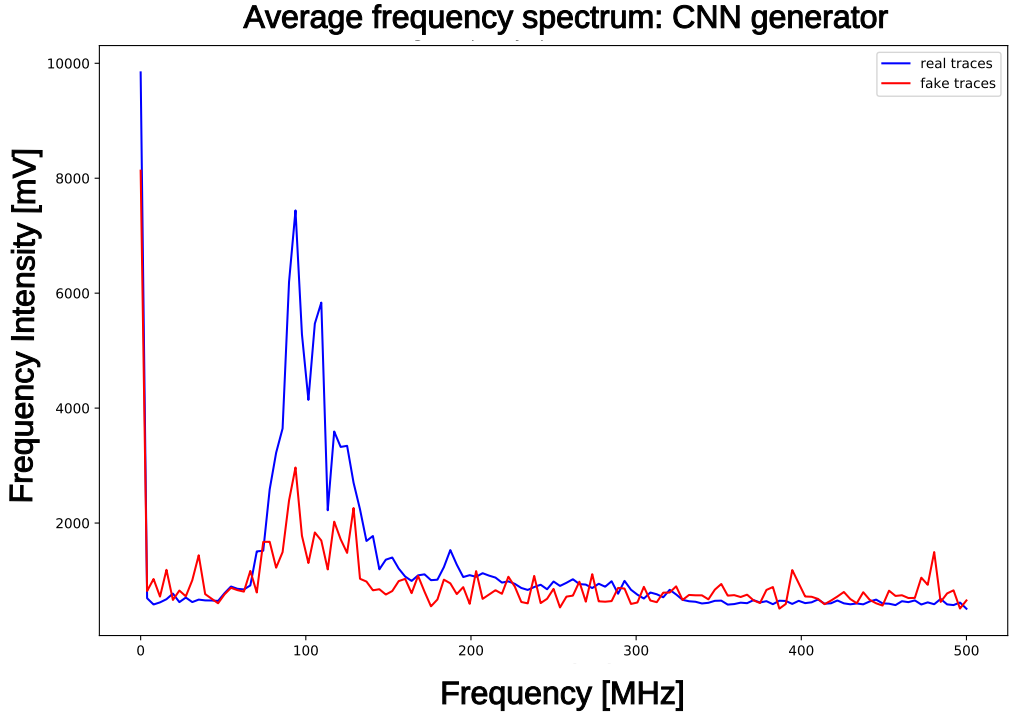

Figure 30: Average frequency spectrum of the CNN generator (blue: real traces; red: synthetic traces)

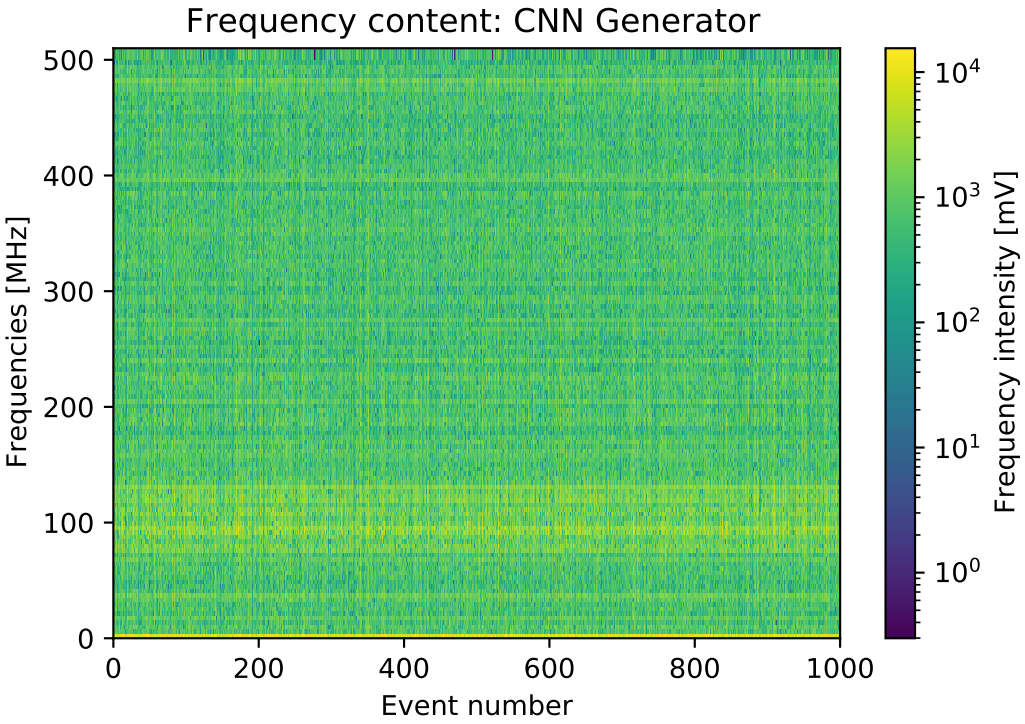

Figure 31: Frequency spectra from different synthesized traces from the CNN generator

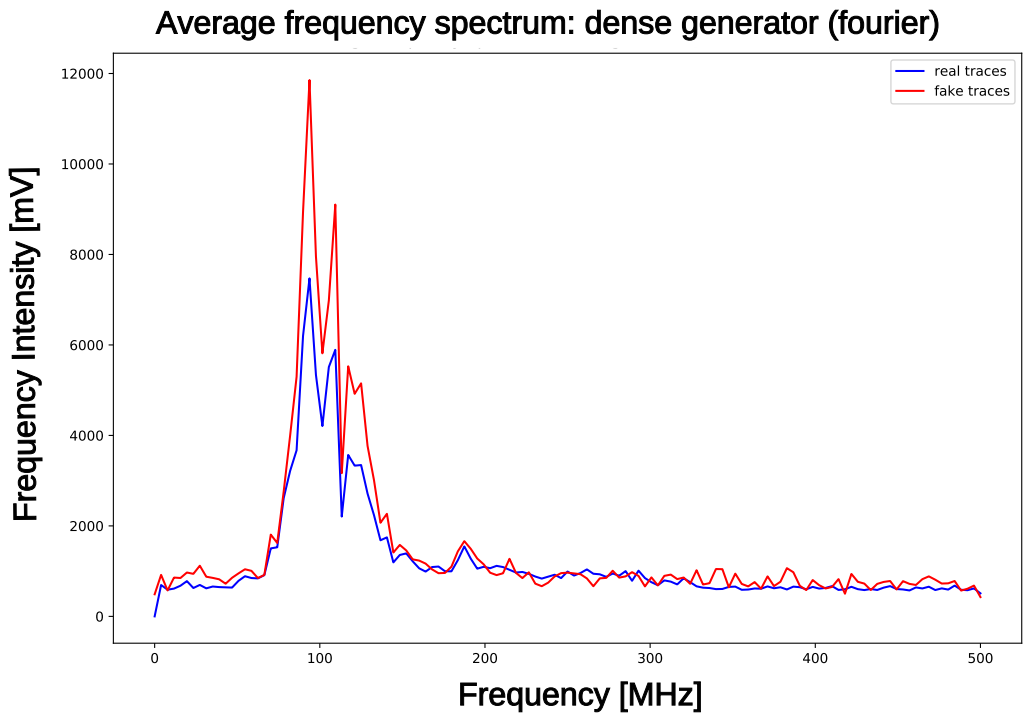

Figure 32: Average frequency spectrum of the Fourier modified dense generator (blue: Fourier modified traces; red: synthetic traces)

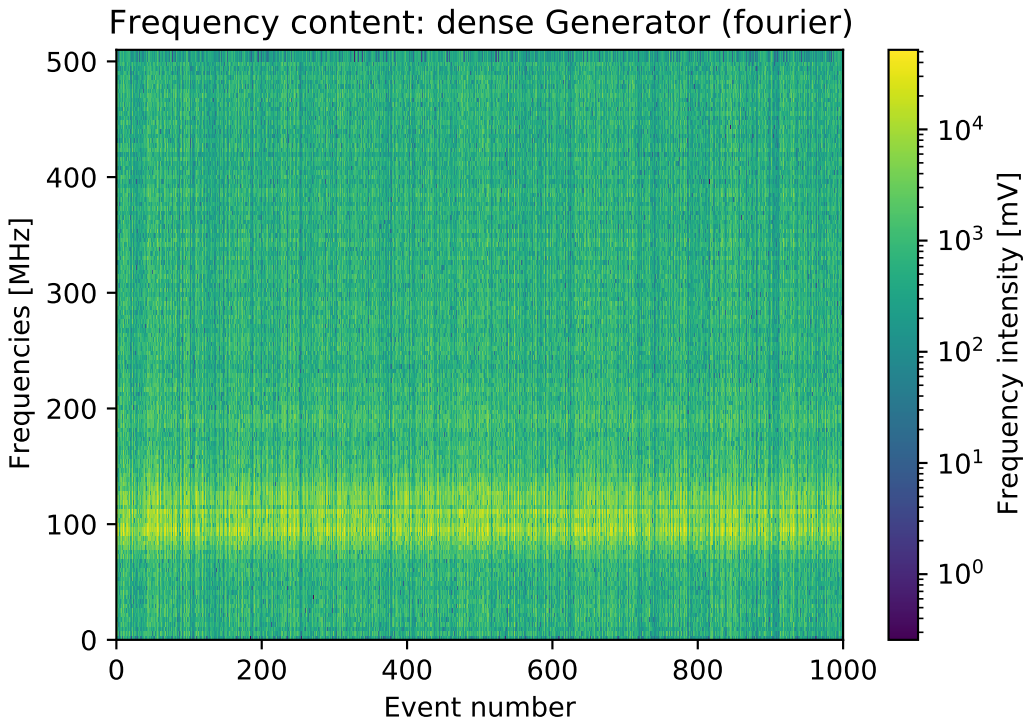

Figure 33: Frequency spectra from different synthesized traces from the Fourier modified dense generator

The average frequency spectrum for traces generated by the dense generator trained on the Fourier modified data set is shown in Figure 32 together with the average spectrum of the Fourier modified data set. One can see that, as expected, there is no DC-component in both spectra. The prominent structure at 100 MHz has a much higher intensity in the generated spectrum than in the real data. The fake spectrum matches the real one past 150 MHz very well.

The sample of generated frequency spectra shown in Figure 33 also show strong similarity with real spectra and the necessary variability in intensity between spectra.

If it were not for the excess intensity of the 100 MHz-structure in the spectrum of the Fourier modified generator, this generator would be the best at reproducing the real frequency spectrum.

4.4 Computational Performance

This section will compare the computational performance of the generator networks to reading in real noise traces. As already mentioned in section 1.1 there are two computational reasons motivating a new method for noise trace generation. Firstly, there is the time aspect. Reading in real data files is time consuming. Reducing this time will help speed up simulations and therefore speed up the analysis process for neutrino

events.

	Computation time [ms]	speed up factor
reading in real data	2.28 ± 0.08	1
dense generator	0.28 ± 0.06	8.14
dense generator (randomized)	0.27 ± 0.08	8.44
dense generator (Fourier modified)	0.25 ± 0.08	9.12
CNN generator	1.44 ± 0.37	1.58

Table 1: Computation time for one noise trace and speed up factor compared to reading in the data

To test the computation speed the time to read in or generate 7068 traces was measured and then divided by the number of traces to to get the time it takes to sample one trace. This process was repeated 10 times and the results of these measurements was averaged. The results of these measurements are shown in Table 1. The measurement uncertainties given are 1-σ uncertainties. The speed up factor in Table 1 was calculated by dividing the time it took to read in the real data by the time it took to perform the process of the corresponding row in the table.
As expected, all generators are faster than reading in real traces. One can also see, that the less complex dense generators are almost six times faster than the CNN generator. A significant time difference between the three dense generators was not found and also not expected because the three generators use the same network architecture.
All the time measurements were performed on a laptop equipped with an Intel Pentium N4200 CPU and no graphical processing unit. After seeing the results from Erdmann et al. [17] one would expect, that the computation times for all the generator networks would reduce further when performing the simulation on a more powerful computer, especially when equipped with a stand alone GPU. Reading in real data on the other hand would not profit from a more powerful computer, which means,that the speed up factors would become even bigger than they are presented here.
This speed up in producing noise will become even more important for analyzing RNO-G events in the future because the traces recorded by RNO-G antennas will be almost ten times longer, containing 2048 instead of 256 time bins, and each event will contain more traces because there are more antennas in an RNO-G-station than the four antennas from ARIANNA.

Secondly, the generator networks will also reduce the demand on the storage capacity necessary to do realistic simulations. At the moment one would have to have multiple gigabytes of noise traces stored on the computer to get a representative sample of noise to add to the event simulations. These large amounts of data also need to be accessible for everyone wanting to perform simulations, which means, that these large files need

to be stored on a server from which they can be downloaded.
Compared to this process the demands of using a generator network are tiny. A trained dense generator requires only 0.564 MB of storage space and the CNN generator 6.47 MB. Using a neural network generator also requires a working tensorflow installation, which adds another 375 MB, but it is still not comparable to the amount of storage space needed using the current method.
Trying to reduce the amount of storage space required with the current method by just using less data, i.e. only read in 1000 noise traces, would result in adding the exact same noise to every 1000th simulation, which is not ideal. The traces generated by the networks are based on non-repeating samples of random numbers and therefore unique.

5 Conclusion

In conclusion the results of this thesis show, that the generator networks were able to learn and reproduce some of the characteristic features of the training data set, but were not capable of learning its entire complexity yet. Both the big potential of the WGAN approach in radio noise simulation and the expected improvement in computational performance were confirmed by this thesis, but further improvements in optimizing the generator are necessary before the WGAN-generator could be used in simulations for data analysis.
Regardless of which steps are performed next I would suggest to use a more powerful computer, equipped with a stand alone GPU, to perform the training, especially when trying to add more layers to the networks. Otherwise the training loop takes hours or in some cases even days to complete, which makes experimenting with different network configurations and training parameters time consuming and impractical. It would be ideal, if training the WGAN could be performed on a large computation cluster. This would remove the demands on ones local machine, which could then be used to work on something else while the training is performed.
With current results the two networks performing best are the dense networks, which were trained on the randomized and on the Fourier modified data set. Randomizing the data set improved on reproducing event to event variability and removing the DC-offset from the data set improved the reproduction of the spectrum. Possible future steps in improving the network performance could be to explore these training methods further. One could either try performing more training iterations on the already trained networks, i.e. use more data for training or training the network for more than 10 epochs. Another approach could be, to train the network on noise traces in frequency space and then transform the synthetic spectra back into time domain. One could also try adding complexity to the networks by incorporating more hidden fully

connected layers. Another way to optimize the training results could be to try and optimize the hyperparameters of the training loop to improve how fast or how well the network converges to its optimal state.
The final version of the training loop can be found at:
`https://github.com/nu-radio/NuRadioMC/tree/arianna_noise_gan/NuRadioReco/modules/io/noise/arianna_noise_gan`

References

[1] J.A. Aguilar et al. "Design and sensitivity of the Radio Neutrino Observatory in Greenland (RNO-G)". In: *Journal of Instrumentation* 16.03 (Mar. 2021), P03025. ISSN: 1748-0221. DOI: 10.1088/1748-0221/16/03/p03025. URL: http://dx.doi.org/10.1088/1748-0221/16/03/P03025.

[2] I. Kravchenko et al. "RICE limits on the diffuse ultrahigh energy neutrino flux". In: *Physical Review D* 73.8 (Apr. 2006). ISSN: 1550-2368. DOI: 10.1103/physrevd.73.082002. URL: http://dx.doi.org/10.1103/PhysRevD.73.082002.

[3] Timothy Miller, Robert Schaefer, and H. Brian Sequeira. "PRIDE (Passive Radio [frequency] Ice Depth Experiment): An instrument to passively measure ice depth from a Europan orbiter using neutrinos". In: *Icarus* 220.2 (Aug. 2012), pp. 877–888. ISSN: 0019-1035. DOI: 10.1016/j.icarus.2012.05.028. URL: http://dx.doi.org/10.1016/j.icarus.2012.05.028.

[4] P. Allison et al. "Design and performance of an interferometric trigger array for radio detection of high-energy neutrinos". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 930 (June 2019), pp. 112–125. ISSN: 0168-9002. DOI: 10.1016/j.nima.2019.01.067. URL: http://dx.doi.org/10.1016/j.nima.2019.01.067.

[5] S. W. Barwick et al. *Design and Performance of the ARIANNA Hexagonal Radio Array Systems*. 2014. arXiv: 1410.7369 [astro-ph.IM].

[6] Kenneth Greisen. "End to the Cosmic-Ray Spectrum?" In: *Phys. Rev. Lett.* 16 (17 Apr. 1966), pp. 748–750. DOI: 10.1103/PhysRevLett.16.748. URL: https://link.aps.org/doi/10.1103/PhysRevLett.16.748.

[7] G. T. Zatsepin and V. A. Kuzmin. "Upper limit of the spectrum of cosmic rays". In: *JETP Lett.* 4 (1966), pp. 78–80.

[8] "Introduction to Machine Learning". In: *Python® Machine Learning*. John Wiley Sons, Ltd, 2019. Chap. 1, pp. 1–18. ISBN: 9781119557500. DOI: https://doi.org/10.1002/9781119557500.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119557500.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119557500.ch1.

[9] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[10] Shiva Verma. *Understanding different Loss Functions for Neural Networks*. accessed 08.07. 2021. URL: https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718.

[11] Antonia Creswell et al. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (Jan. 2018), pp. 53–65. ISSN: 1053-5888. DOI: 10.1109/msp.2017.2765202. URL: http://dx.doi.org/10.1109/MSP.2017.2765202.

[12] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[13] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[15] Cédric Villani. *Optimal Transport*. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-540-71050-9. URL: https://doi.org/10.1007/978-3-540-71050-9.

[16] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. "Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters". In: *Physical Review Letters* 120.4 (Jan. 2018). ISSN: 1079-7114. DOI: 10.1103/physrevlett.120.042003. URL: http://dx.doi.org/10.1103/PhysRevLett.120.042003.

[17] Martin Erdmann, Jonas Glombitza, and Thorben Quast. "Precise Simulation of Electromagnetic Calorimeter Showers Using a Wasserstein Generative Adversarial Network". In: *Computing and Software for Big Science* 3.1 (Jan. 2019). ISSN: 2510-2044. DOI: 10.1007/s41781-018-0019-7. URL: http://dx.doi.org/10.1007/s41781-018-0019-7.

[18] *VISPA cluster: Deep Learning examples*. accessed 15.07.2021. URL: https://vispa.physik.rwth-aachen.de/server/?workspace:VISPA%20Cluster/examples:Examples:.

[19] Y Rath J.Glombitza. *Deep Learning Tutorial: Generative Adversarial Networks*. accessed 15.07.2021. Feb. 19, 2018. URL: https://indico.scc.kit.edu/event/344/contributions/2425/attachments/1248/1749/GAN_tutorial_hap_workshop.pdf.