Department of Physics and Astronomy
University of Heidelberg

Master's thesis in Physics submitted by
Florian Ernst

born in
Wermelskirchen, Germany

2023

Exploring possibilities for higher dimensional calorimeter simulations

This Master thesis has been carried out by
Florian Ernst

at the
Institute of Theoretical Physics

under the supervision of
Prof. Tilman Plehn

Contents

1 Abstract

This Master's thesis describes several approaches to generate calorimeter simulations with uncertainty estimation using machine learning. The main focus is on normalizing flows and variational autoencoder. We analyze the corresponding advantages and disadvantages of these models, paying strong attention to their generation quality and their scaling with respect to larger datasets. For the second point we analyze the behavior of these models, when applying them to different datasets with increasing dimensionality. We find that the pure normalizing flow is consistently superior to the variational autoencoder based approaches. However, it needs more parameters for the same dataset, what makes it impossible to apply for large datasets. The variational autoencoder on the other hand, is able to handle large input dimensionalities without performing significantly worse. We especially introduce a kernel variational autoencoders, a compromise between a fully connected and a convolutional approach, that is able to handle our largest dataset with a dimensionaliy of 40500 features.

2 Zusammenfassung

Diese Masterarbeit beschreibt verschiedene Ansätze um schnelle Kalorimetersimulationen mit Fehlerabschätzung durch maschinelles Lernen zu generieren. Dabei liegt das Hauptaugenmerk auf "normalizing flows" und "variational autoencoders". Wir analysieren die jeweiligen Vor- und Nachteile dieser KI-Modelle, mit einem Fokus auf die Präzision ihrer Vorhersagen und ihre Skalierungseigenschaften bezüglich der Datendimensionalität. Für die Analyse des zweiten Punktes untersuchen wir das Verhalten der Modelle, wenn sie auf verschiedenen Datensätzen mit steigender Dimensionalität trainiert werden. Dabei stellen wir fest, das ein separater normalizing flow einem Ansatz mit variational autoencoder in der qualität grundsätzlich überlegen ist. Allerdings braucht der normalizing flow mehr Parameter, was dazu führt dass er nicht bei großen Datensätzen verwendbar ist. Der variational autoencoder hingegen, ist in der Lage diese Datensätze zu bewältigen ohne dabei seine Qualität weiter einschränken zu müssen. Insbesondere stellen wir den von uns entwickelten "kernel variational autoencoder" vor, ein Kompromiss zwischen einem "fully connected" und einem "convolutional" KI-Modell. Dieses Modell ist in der Lage selbst unseren größten Datensatz, mit einer Dimensionalität von 40500 features, zu verarbeiten.

3 Theory

3.1 Particle Physics

In order to understand the need for faster detector simulations with uncertainty estimates one has to understand this task in a larger context — namely in the context of the standard model of particle physics. They are an essential tool to probe this theory.

3.1.1 Standard Model

The standard model employs quantum field theory to describe the behavior of subatomic particles. It does this by assuming a Lagrangian $\mathcal{L}$ to be invariant under a specific group transformation. For the standard model the group $SU(3) \times SU(2) \times U(1)$ is used, as nature is apparently described by it. It enables us to describe the electroweak interaction $(SU(2) \times U(1))$ and the strong interaction $(SU(3))$. A large scale analysis of the mathematics behind the standard model is beyond the scope of this introduction and we refer to the corresponding literature. [1–11]
On a phenomenological level, the standard model describes the interaction between the matter particles (fermions) by introducing gauge bosons that carry the forces between the individual particles. A systematic overview of the individual fermions and gauge bosons can be seen in Figure 1. The standard model is currently one of the most accurate physical models in general and describes our world with high accuracy. In the rest of this section we are summarizing the standard model as it is described in [12].

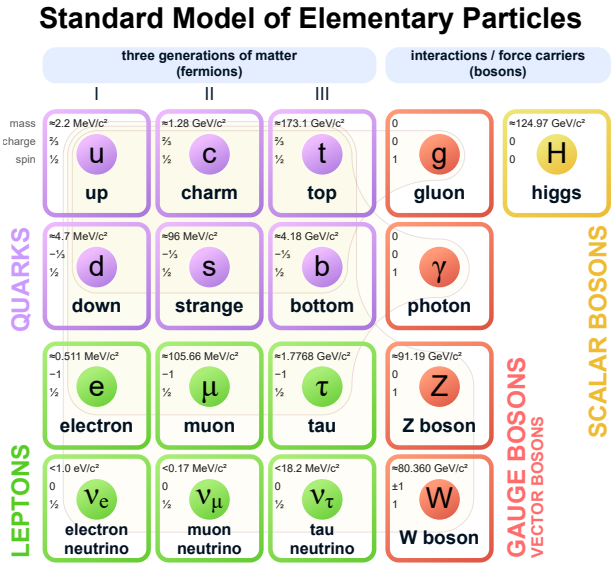

Figure 1: Visualization of the individual elementary particles of the standard model. Credits for this figure to [13].

3.1.2 Gauge bosons

Gauge bosons are the mediators of the three fundamental forces and carry a spin of 1. The photon γ, one of them, mediates the electromagnetic interaction and corresponds to an abelian $U(1)$ symmetry. Every particle that carries an electric charge interacts with photons and thus electromagnetically.
The three weak bosons W^+, W^- and Z are mediating the weak interaction, responsible for particle decays. These are associated to a $SU(2)$ symmetry. The first two $W^\pm$ are interacting with the photon, while the Z-boson is not, as it is neutrally charged. A salient feature of the weak interaction is its exclusive coupling to left-handed chiral states of fermions. In the context of the weak interaction, fermions carry a quantum number known as weak isospin (T). Specifically, left-handed fermions are grouped into doublets with weak isospin $T = \frac{1}{2}$, where the third component T_3 distinguishes the members of the doublet. For instance, in the case of leptons, the electron-neutrino and the electron form such a doublet with $T_3 = +\frac{1}{2}$ and $T_3 = -\frac{1}{2}$, respectively. The $W^\pm$ bosons couple to transitions between these doublet members, whereas the Z boson couples to both members without changing T_3.
The gluon g is the mediator of the strongly interaction, also known as strong nuclear force. This force is the main reason of why protons and neutrons are held together. Only particles that carry a color charge, the charge of the strong force, are interacting strongly. In nature we do not find any free particles with a color charge, instead these particles are bound in color-neutral so-called "hadrons". This is called "confinement". The reason is that the energy, needed to create a new particle, is smaller than the energy to create a new strong interacting fermion. Therefore, if a strongly interacting fermion is be separated from a hadron, new fermions arise in between the separated fermion and the hadron. This way a chain of hadrons is created, that is called "jet", resuling in serveral bound hadrons, but no free strongly interacting fermions.

3.1.3 Fermions

The fermions are the other half of the elementary particles, described by the standard model. They are classified in two general categories: "leptons" and "quarks". Furthermore, leptons are divided into electric charged particles and corresponding neutrally charged "neutrinos". All fermions carry a spin of $\frac{1}{2}$ and all left-handed fermions interact via the weak interaction. Except for neutrinos, which are not carrying an electric charge, all fermions are also interacting electromagnetically. Quarks are special as they also carry a color charge and thus interact strongly.
Within each of these three classes (neutrinos, charged leptons and quarks) three "generations" exist. Particles in one generation (usually) have a similar mass. The exception to this rule are the neutrinos. They are almost massless and we only know that they have a mass, but not the size of it [14]. However, in the Standard Model (and in this section) they are usually considered as "massless" particles, even though the addition of neutrino masses is straight forward in this theory framework. As the mass increases with the generation, only the first generation of the massive particles is stable.
The four most important fermions, for this thesis, are the up (u) and down (d) quark, the electron (e^-) and the muon (μ^-). the u-quark is a first generation quark with an electric

charge of $\frac{2}{3}$. The d-quark is the other first generation quark with an electric charge of $-\frac{1}{3}$. The electron is the first generation charged lepton with a charge -1, the muon is the second generation charged lepton with the same charge. Even though the muon is not stable, it has significant impact on the experimental setup of modern particle physics experiments.
For every elementary particle, there exists a corresponding anti-particle with an opposite sign for all charges, but the same mass. For example the positron (e^+) is the anti-particle of the electron. For leptons, we change the superscript of the charge, like for the positron, for other particles we add a bar over the name. E.g. the anti-down-quark is noted as $\bar{d}$.

3.1.4 Higgs boson

There is one additional particle in the standard model, that was not mentioned before. It is a scalar boson and called the "higgs boson" (H), carrying a spin of 0. This boson arises from the "electroweak symmetry breaking" and it introduces masses for all the other elementary particles via the "Higgs-mechanism". One important effect of the electroweak symmetry breaking is that the $U(1)$ symmetry of the γ is not the $U(1)$ summetry in the $U(1) \times SU(2)$ electroweak symmetry group. Instead it is an unbroken diagonal subgroup of the broken original $U(1) \times SU(2)$ symmetry group. [15–18]

3.1.5 Hadrons

Hadrons are, strictly speaking, no elementary particles. Nevertheless, they are strongly correlated to the quarks and the concept of confinement.
Hadrons are bound states of two or three quarks. They are classified into mesons, which are bound states of a quark and an anti-quark, and (anti-)baryons, which consist of three (anti-)quarks.
For this thesis, especially, the pion (π), a meson consisting of combinations of the first generation quarks and anti-quarks, is important. There are three types of pions: The π^+ ($u\bar{d}$), the π^- ($d\bar{u}$) and the π^0 ($\frac{1}{\sqrt{2}}(u\bar{u} + d\bar{d})$).

This was only a very short overview over the full standard model as it is needed to understand the the following chapters of this thesis. In fact, there exists a full particle-zoo arising from its particles and many important concepts of particle physics were omitted. We refer the interested reader to the corresponding literature [11, 12].

3.2 Detector physics

Particle detectors are devices that we need to verify the predictions of the standard model and to infer the free parameters in it, for example the particles' masses. Usually they are used in combination with a particle collider.
The main idea is to collide two particles at some energy level E (collider) and to measure the energies that the resulting particles deposit in the detector. This means, that it is necessary to understand how particles lose energy in matter in order to understand particle detectors.

3.2.1 Particles in matter

The energy loss in matter is different for each particle. This individual footprint enables us to distinguish different particle detections. The individual effects are, again, summarized from [12].

Charged leptons Charged leptons lose energy in matter mainly due to two different reasons:

- Ionization
- Bremsstrahlung

Ionization is the effect, where the lepton removes an electron from an atomic shell of an absorber atom, ionizing the corresponding atom. The energy loss by ionization satisfies the "Bethe-Bloch" formula [12, 19], which states that the energy loss depends on the absorber material mainly by its density ρ, as the mass number A and the atomic number Z are almost proportional for stable atoms:

$$\frac{\mathrm{d}E}{\mathrm{d}x} \approx -4\pi\hbar^2 c^2 \alpha^2 \frac{\rho Z}{m_\mathrm{e} m_\mathrm{u} A \mathrm{v}^2} \left\{ \ln\left[\frac{2\beta^2\gamma^2 c^2 m_\mathrm{e}}{I_\mathrm{e}} \right] - \beta^2 \right\}$$

Here, h is Planck's constant, c the speed of light, m_e the mass of the electron, m_u the mass of a proton and α the electromagnetic coupling strength. A and Z are the mass number and the atomic number, respectively, of the absorber material and I_e is the effective potential averaged over all absorber electrons. β and γ are the corresponding Lorentz transformation factors and v is the velocity of the ionizing particle.
Particles with $\beta\gamma \approx 3$ are called "minimal ionizing" as they are close to the minimum of the Bethe-Bloch formula. These particles do not lose much energy because of ionization processes.
Bremsstrahlung is the process of the lepton ℓ emitting a photon in the electric field of another atom $\ell \to \ell + \gamma$. The strength of this effect is proportional to the inverse mass of the lepton $r \propto \left(\frac{1}{m_\ell}\right)^2$. Bremsstrahlung happens only above a critical energy $E_c \approx \frac{800}{Z}$ MeV.
In a modern detector Bremsstrahlung is the dominant effect for electrons, but ionization is dominant for muons, as they are much heavier than electrons. As muons are usually produced close to $\beta\gamma = 3$, they are penetrating large parts of the detector, before they are stopped by ionization - usually outside of the detector.

Photons There are three processes, that can be dominant for the photon energy loss:

- Photoelectric effect
- Compton scattering
- e^+e^- pair-production

The photoelectric effect describes the photon being absorbed by a shell electron, increasing the electrons energy: $\gamma + e^- \to e^-$. However, this process happens usually for energies below the MeV range and is not very relevant for collider experiments.
Compton scattering describes the scattering of the photon off a shell electron, $\gamma + e^- \to \gamma + e^-$. Also this process is not reducing the photon's energy significantly in our current setup.
In a collider experiment, the main reason for the photon to reduce its energy is pair production. Therefore, the photon decays into an electron-positron pair $\gamma \to e^+ + e^-$. This effect is dominant when $E_\gamma > 10$ MeV and each particle is carrying an energy of $E_{new} = \frac{E_\gamma - m_e}{2} \approx \frac{E_\gamma}{2}$. Therefore, this effect vanishes entirely, if the energy of the photon is smaller than the rest mass of two electrons.

Electromagnetic shower Now we see that the dominant energy loss processes of the electron and the photon are cyclic. A photon decays into an electron and a positron. These two create two additional photons, which decay again - an exponential growth in the number of particles. This cascade of particles is called "electromagnetic shower" (cf. Figure 2). The distance between two splittings, X_0, can be approximated by $\frac{7}{9}$ of the mean free path length of the pair production process. This results in values for X_0 of $O(1)$ cm. As the energy loss is exponential in an electromagnetic shower, there are not many generations of splittings in one shower, such that the overall size of those showers is in the regime of dm.
Since the electromagnetic shower consists of only three different particles and since it produces a large number of particles in total, the overall fluctuations for showers of the same incident energy are small. The shower ends once the critical energy for pair creation is reached.

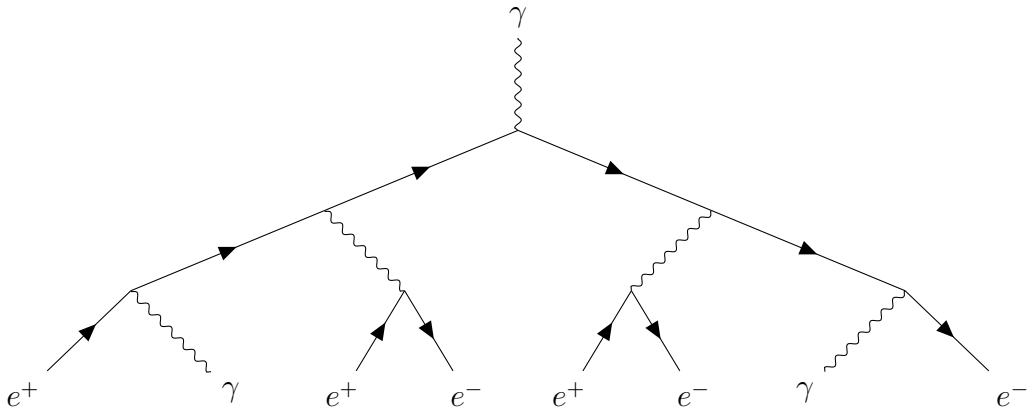

Figure 2: Visualization of an electromagnetic shower.

Hadrons Hadrons have an additional energy loss mechanism. Of course, all the possibilities of the charged leptons exist, but hadrons can also interact strongly with the protons and neutrons of the absorber material. If their energy is larger than ≈ 300 MeV they are able to create other hadrons (the lightest hadron is the pion with $m_\pi \approx 140$ MeV). At this point the hadronic energy loss is dominated by hadronic showers. They are conceptually similar to electromagnetic showers but way more diverse as they contain all hadrons that

are light enough to be produced. This results in larger fluctuations than for electromagnetic showers. In fact, as $\pi^0 \to \gamma + \gamma$, every hadronic shower contains electromagnetic showers as a component.
Also the average size of a hadronic shower is different to the compact electromagnetic shower. Typically, the distance between two splittings for hadrons, the "nuclear interaction range" λ_I, is already in the regime of decimeter. Resulting in showers that stretch over up to several meters.

3.2.2 Detector layout

To ensure that all the particles, that arise after the collision, can be detected, all these energy loss mechanism have to be considered when constructing a particle detector. These restrictions resulted in a cylindrical characteristic sandwich-structure that most modern detectors implement. This cylinder is orientated such that its rotational axis corresponds to the colliding particles' beam. As coordinates, the variables η and ϕ are chosen. ϕ is the azimuth angle corresponding to the cylinder, $\eta = \ln\left[\tan\left(\frac{\theta}{2}\right)\right]$ is the "pseudo-rapidity". The variable θ is the angle between the beam axis and the momentum axis of the individual particle, the polar angle [20].
Typically, the inner part of the detector consists of "tracking chambers" that measure the bending of the particle's trajectories in a magnetic field. In this first ionization layer, the energy deposition is less relevant. However the bending allows for a momentum measurement of the particles. Afterwards an "electromagnetic calorimeter" is used to measure the energies of electrons and photons. Typically, it is of the size as the expected electromagnetic showers. The third layer is a "hadronic calorimeter". At this distance from the collision point, the electrons and photons have already been stopped and the energy of the resulting hadronic jets is measured. Because of the large size of the hadronic showers, also the hadronic calorimeter has to be large. Often it is the largest segment of a particle detector. The outermost part of a usual particle detector are the muon chambers. These are needed to detect muons that are released during the collision. Since these muons are highly penetrating, they are the only particles, besides neutrinos, that reach the outermost part of the detector [12, 21]. A schematic visualization of the ATLAS detector can be seen in Figure 3.

Tracking chamber The tracking chamber is used to measure the momentum of the corresponding charged particles. The idea is to measure the curvature of their trajectories in a magnetic field. From electrodynamics, we know that the curvature of the bending is determined by the particles' charge q, the magnetic field strength $\vec{B}$ and the momentum $\vec{p}$. In fact, the important quantity is $\vec{p} \times \vec{B}$, so one has to ensure that the particle trajectory is orthogonal to the B-field.
Together with the energy measurement in the later calorimeters, this enables us to reconstruct the full momentum four-vector and thus the mass of the detected particle.
In practice either gas chambers of silicon tracing detectors are used. Gas chambers use a gas that is ionized by the individual particles. The current, arising by freeing the electrons, can be detected and used to infer the particle trajectory. Silicon trackers rely on the creation

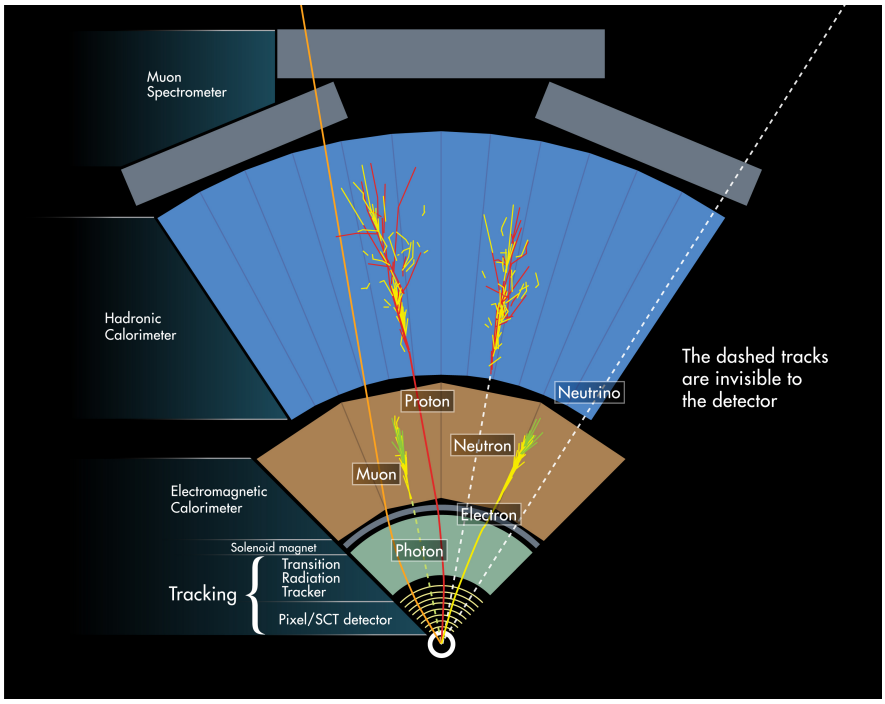

Figure 3: Visualization of the ATLAS detector. Image taken from [22].

of electron-hole pairs in silicon semiconductors. To measure the trajectory of the particle in question, a matrix of semiconductors is used. If the particle traverses through one of these semiconductors, the electron-hole pairs create a measurable current that is used to identify the particles positions [12, 23].

Electromagnetic calorimeter The electronic calorimeter is used to measure the energy of the corresponding electromagnetic showers. The problem with calorimeters is that they need a high-Z material, like lead, to initiate the shower and an active material that actually measures the energy. The latter usually happens via the detection of scintillation light. Two types of calorimeters are distinguished. The first type is called "sampling calorimeter". It uses an alternating sandwich-setup of an absorber layer and an active layer constructed by a different material. The shower starts in the the absorber but only the energy that is deposited in the active layer can be measured. The alternative is to use a "homogeneous calorimeter". This type uses the same material as absorber and active material, e.g. lead tungstate ($PbWO_4$). This results in better energy resolutions, as all the energy can be measured, but usually also in larger calorimeter sizes. Due to the worse shower properties, the showers are typically larger in homogeneous calorimeters. Furthermore, the are more expensive than sampling calorimeters.
In the end, several cells of the active material are used to achieve spatial resolution. However, this resolution is usually not very high and inferior to the tracking chamber [24].

Hadronic calorimeter The hadronic calorimeter is usually constructed as a sampling calorimeter. The fact that it is already the largest part of the detector increases the need to make it as compact as possible. Furthermore, it is more money-efficient to build it as a sampling calorimeter. The corresponding materials are cheaper [25].

3.2.3 Detector Simulations

Now, that larger context is explained, it becomes obvious, that detector simulations are one of the most important parts of the analysis of a collider experiment. The interaction of a particle with the individual detector materials is understood very well, we are able to predict the energy depositions of, for example an electron, very accurately given the detector and the particle details. However, this prediction is not invertible: We cannot predict all particle details exactly from the energy depositions. Reasons are mainly the finite resolutions of our detector and noise - detector effects. The solution is to predict a set of measurements from our theory and compare it statistically with our actual set of measurements. The downside of this inference step is that we need a large set of predictions to make the comparison accurate. The interactions of the particles with the detector are usually simulated using the tool GEANT4 [26–28]. This state-of-the-art software uses Monte-Carlo simulations to predict the energies that are deposited in the corresponding cells of our calorimeters. However, as the number of particles in a shower is very large, these simulations of the calorimeters on particle level are expensive (cf Figure 1 in [29]). Especially for high incident energies, where many particles are produced, the run-time of the detector simulation tools is problematic. In future runs of current colliders, like the LHC, this time consuming simulation is expected to be a bottleneck of the overall evaluation. Therefore, the particle physics community tries to find alternative solutions to prevent this by establishing new algorithms to generate detector simulations using machine learning [30–40]. One of these efforts is the CaloChallenge [41]. This is a competition to find the "best" method for fast and accurate calorimeter simulations, building the foundation for future collider experiments. The main idea is to replace the slow Monte-Carlo approach in GEANT4 with a fast machine learning alternative. This thesis is the summary of my participation in this domain. Now, before more details about the challenge are presented, a small overview over the used machine learning techniques will be given. The structure is closely following [42]. An alternative overview over ML can be found in [43].

3.3 Machine Learning

3.3.1 General idea

Machine learning (ML) can be seen as a fit. We try to optimize the parameters θ of a function $f_\theta(x)$ such that it is approximating another (usually unknown) function $f(x)$. However, while classical fits are used on functions with $O(10)$ parameters, ML tries to optimize functions with $O\left(10^6\right)$ or more parameters.
Nevertheless, the process is conceptually similar. Since we do not have a closed form solution for the global optimum of our function parameters, we try to find this optimum iteratively using "gradient descent". Therefore, we use a differentiable function $\mathcal{L}(\theta|TS)$, called "loss function", that describes the "quality" of our fit. Here we call TS the set of points, where we

evaluate the loss function on, our training set. We can find a minimum of the loss function, and thus an optimum of our fit, by shifting the parameters θ in the direction of the negative gradient:

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \mathcal{L}$$

The parameter α is called the "learning rate" and it describes, how fast one is following the gradients of the loss function. One can either use a constant learning rate, or adapt it during the training. This adaption is called "LR-schedule" and is able to improve the convergence behavior. In this thesis we are using either a constant LR or a "one-cycle-LR" [44]. The idea of the one-cycle-LR is to start with a medium-sized LR to let the network find a stable initialization, and increase the LR soon to its maximum value to speed up the optimization. In a third phase, the the LR is slowly decaying to a very low value in the final epoch to find the best configuration in the final optimum.
In practice one usually uses not the full training set to estimate the gradients, but only a subset from it, a batch. This prevents the optimization to get stuck in bad local optima and reduces the needed memory during the optimization.

3.3.2 ADAM

In this thesis we are using a more modern optimization algorithm called "ADAM" [45]. It is minimizing the loss function in analogy to a ball rolling down a hill, by assigning a momentum to the gradients. This enables the optimizer to leave shallow minima, if the parameters were reduced a lot beforehand. Just like a ball that can roll up a small hill, if it has a lot of speed. Empirically, this optimizer is not only converging to a better minimum, but also reaches it faster:

$$p \leftarrow \beta \cdot p + (1 - \beta) \cdot \nabla_\theta \mathcal{L}$$

$$\hat{p} = \frac{p}{1 - \beta^{\#Update}}$$

$$v \leftarrow \gamma \cdot v + (1 - \gamma) \cdot (\nabla_\theta \mathcal{L})^2$$

$$\hat{v} = \frac{v}{1 - \gamma^{\#Update}}$$

$$\theta \leftarrow \theta - \alpha \cdot \frac{\hat{p}}{\sqrt{\hat{v} + \epsilon}}$$

p is the momentum estimate that is generated using a running average, v is a variance estimate to make the convergence more stable and ϵ a small regularization. During the initial updates v and p are corrected as the running average is not trustworthy there. β and γ are two hyperparameters that somehow describe the strength of the friction, the ball is experiencing. We were always using their default values $\beta = 0.9$ and $\gamma = 0.999$.
The actual differentiation, the computation of the gradients, is straightforward by using the chain rule. In fact, the calculation of the gradients is done completely autonomous by common ML-libraries, like PyTorch [46], using "backpropagation" [47].

3.3.3 Neural Networks

Now, that it is clear how to optimize a loss function, we should discuss which function we want to fit. Usually the main building block for the fitted function in ML are "neural networks", short NNs [48–50]. These functions are usually called "models" in the context of ML and are concatenations of linear functions, "linear layers", and non-linear "activation" functions. The latter is needed to model non-linear dependencies.

$$x_i^{(n)} = \mathrm{NL}\left(W_{i,j}^{(n)} \cdot x_j^{(n-1)} + b_i^{(n)}\right).$$

Here, (n) is the index of the corresponding layer of our network and NL symbolizes the (usually non-learnable) non-linearity. In this thesis we will use three types of activation functions. In the inner layers we are going to use the so called "Rectifying Linear Unit", short ReLU

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

or the "leaky ReLU"

$$\mathrm{LReLU}(x; s) = \max(0, x) + s \cdot \min(0, x).$$

In the final layer we are mainly going to use the sigmoid function $\sigma(x)$, since we usually require our output to be between zero and one.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The output dimensionality of the linear layers is called the number of "neurons" of this layer. A schematic overview of a full NN can be seen in Figure 4.
By theory, we know that a sufficiently wide NN, i.e. a NN with enough neurons, with at least two linear layers can approximate any continuous function on a compact set [51, 52]. Thus, we can use this building block for basically any task. Furthermore, one can show that the number of linear regions, a measure of expressivity, of a NN scales at least polynomial with the number of neurons, and at least exponential with the number of linear layers [52]. Thus, a wide network is usually less expressive than a deep network.
Even though NNs are the most important building block of ML models, there are also other options. One of them is the so called "convolutional neural network" (CNN) [53]. It uses $C \cdot C'$ ("channels") learnable "convolution kernels" $W_{c,c'}$ of size $k_1 \times k_2$ "pixels" that are moving over some 2 dimensional input instead of a "fully connected" linear layer. These models are designed and optimized for image-like data. Their scaling behavior is constant with the input dimensionality, while normal NNs scale (at least) linearly. The value of a pixel after the convolution layer is computed the following way:

$$x_{c,i,j}^{(n)} = \sum_{\substack{a \in \{0,\dots,k_1-1\} \\ b \in \{0,\dots,k_2-1\} \\ c' \in \{0,\dots,C'\}}} \left(W_{c,c',a,b}^{(n)} \cdot x_{c',i+a,j+b}^{(n-1)} + b_{c,c'}^{(n)}\right).$$

For the boundary cases usually zeros are appended. The indices i and j after the convolution are not necessary consecutive. One could also chose $i \in \{0, 2, 4, ..\}$ — this spacing between

the new indices is called "stride".
In addition to convolutional layers also pooling layers are used. These layers are shifted over the input image just like the convolutional kernels, but their weights are fixed. In this thesis only average pooling layers ([43], p.13 ff) are used. They correspond to convolutional layers with only one weight matrix that is channelwise applied to all incoming channels. For all weights W the normalization for the average $\frac{1}{k_1 \cdot k_2}$ is chosen.

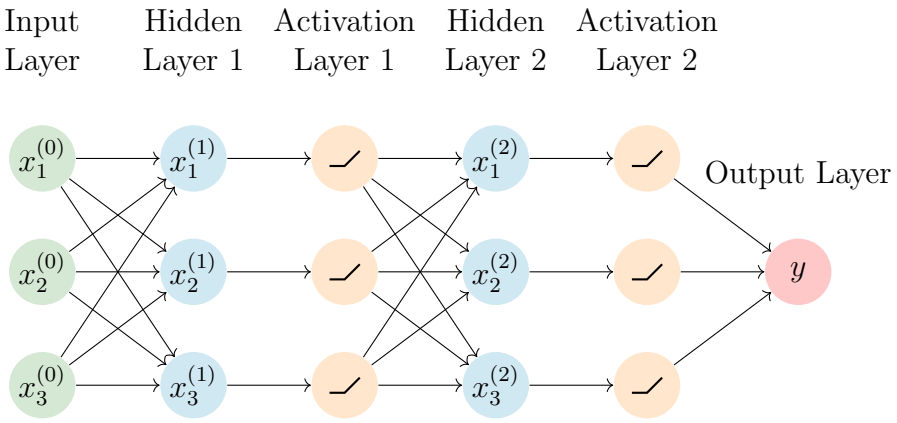

Figure 4: A schematic visualization of a simple neural network.

3.3.4 Classification

Since the concept of NNs and their optimization is clear now, we should discuss the most crucial part of the training algorithm:
How to *find* a good loss function?

The loss function is usually directly connected to the task that we want the model to perform. Thus, the loss function is the only part of ML that truly varies a lot. A usual "hello world" of ML is the optimization of a model to distinguish two different things. This task is generally called "classification" and the corresponding model is a "classifier". More precisely, we try to make our model learn the probability of a point to belong to a class C.
In order to train this classifier, we are going to use the KL divergence between two probability distributions. The true probability of one point x to belong to either class 0 ($p_0(x)$) or class 1 ($p_1(x) = 1 - p_0(x)$) and the corresponding probability $\hat{p}_i$ as predicted by the network. This is justified by the Neyman-Pearson lemma, which states the likelihood ratio is the most powerful test statistic to distinguish two hypotheses [42, 54].

$$\begin{aligned}
\mathcal{L}_{\text{class}} &= \sum_{j=0,1} D_{\text{KL}}\left[p_j, \hat{p}_j\right] \\
&= \left\langle \log p_0 - \log \hat{p}_0 \right\rangle_{p_0} + \left\langle \log p_1 - \log \hat{p}_1 \right\rangle_{p_1} \\
&= -\left\langle \log \hat{p}_0 \right\rangle_{p_0} - \left\langle \log \hat{p}_1 \right\rangle_{p_0} + \text{const}(\theta)
\end{aligned}$$

The constant term in the loss function can be ignored, as it cannot influence our gradients during the optimization.

Now, we can approximate the full KL-divergence, over all possible points, by the sum over the TS. In this step, we are assuming that the TS is a good representation of the possible output of the function that we want to model.

$$\begin{aligned} \Rightarrow \quad \mathcal{L}_{\text{class}} &\approx -\sum_{x \in \text{TS}} \left[p_0(x) \log \hat{p}_0(x) + p_1(x) \log \hat{p}_1(x) \right] \\ &= -\sum_{x \in \text{TS}} \log \left[\hat{p}_1(x)^{p_1(x)} \cdot (1 - \hat{p}_1(x))^{(1-p_1(x))} \right] \end{aligned}$$

Thus, we receive the cross entropy between the true probability and the predicted probability as our loss function.

In practice one finds that the classifier is modeling the likelihood ratio very accurately. Because of this, its predictions can be used to quantify the quality of the predictions of other (generative) models [39, 55, 56]. In this thesis two distinct measures are used to evaluate the classifier's judgement: The "AUC" of its "ROC curve" and the weight distributions.

ROC curve A classifier is used to separate two types of samples. However, it is, in fact, predicting more than a label, but a probability to belong to a class. Therefore, it is not necessary to apply the decision boundary at 50%. For example, one can also decide that a chance of 10% for class 0 is still good enough to be classified as class 0, if the cost of a false classification as class 1 is high enough. For now, let's assume, that we want to identify a member from class 1 out of a set of class 0 samples. Then, we can define the number of samples correctly classified as class 1, $N_{\text{True positive}}$, and the number of samples from class 0, that are falsely classified as class 1, $N_{\text{False positive}}$. Furthermore, lets label the number of all samples from class 0 N_0 and the number of samples from class 1 N_1. Using this values as functions of the decision boundary d, one can define the ROC curve (receiver operating characteristic curve) the following way:
For each decision boundary d, compute the true signal rate

$$\epsilon_S(d) = \frac{N_{\text{True positive}}(t)}{N_1}$$

and the background rate

$$\epsilon_B(d) = \frac{N_{\text{False positive}}(t)}{N_0}.$$

Than, plot the curve consisting of the tuples $(\epsilon_S(d), \epsilon_B(d))$, the ROC curve. This curve describes the ability of the classifier to separate the two classes. A ROC curve that is the diagonal corresponds to random guessing and we assume that our model cannot perform worse than this. Thus, the larger the area between the diagonal and the actual ROC curve, the better the classifier. Thus, the area under the ROC curve, the AUC, is a single number that is able to describe the separation power of the classifier. This means, in the context of a classifier test, that if an optimal classifier has a bad ROC curce, or an AUC close to 0.5, the fake samples are indistinguishable from the real samples.

Weight distributions The second measure are the probabilities that the classifier predicts. They correspond to the probability $p(x)$ that a sample x is generated by an artificial method. Now, it is possible to compute the "weight" $w(x) = \frac{p(x)}{1-p(x)}$ of this sample. For similar, almost indistinguishable, sets of real and fake samples, we expect the weight distributions of both subsets to peak at 1. Furthermore, the true weights should reach to higher weight values, while the fake weights should reach to lower values. These weights can be used to reweight histograms by considering that $w(x) = \frac{p_1(x)}{p_2(x)}$, where p_1 is the generative model's predicted probability and p_2 is the true probability. Therefore, the weight can be used to correct the probability predictions of a generative model. Additionally, the weights can be used as a filter criterion for some high level observables, enabling us to find failure modes in the analyzed other model.

3.3.5 Invertible Neural Networks

One of the two most important architectures used in this thesis are invertible neural networks (INN) or normalizing flows [57–59]. In fact INNs are a special case of normalizing flows. Other cases are listed in section 3.3.5, "Other flow types". These models are used for density estimation and generation tasks. We use them primarily for the latter, as we want to generate realistic detector simulations as explained in subsubsection 3.2.3. The idea behind INNs is to morph an arbitrary probability distribution into a "latent distribution", usually a normal distribution, by using an invertible function. Of course this is not possible in general, as we need to preserve the topology if we want to use a bijection, which is necessarily a homeomorphism since our network (and its inverse) is differentiable [60]. In fact, this is one of the biggest downsides of INNs. However, let's assume for now, that the topology of the modeled distribution is the same as the topology of the latent distribution. We will come back to this problem later in section 3.3.5, "Final Comments".
If we are able to find this invertible transformation, we can sample a point from the standard normal and map this point into the space of our training set. Effectively enabling us to sample from the underlying probability distribution of the training set. This makes it possible to generate something new by just observing existing samples, so we are trying to solve a "generative" problem.
To derive the loss function of an INN we are using the change of variables formula:

$$\begin{aligned} & dx\ p_{\text{model}}\left(x \mid \theta\right) = dz\ p_{\text{latent}}\left(z\right) \\ \Leftrightarrow \quad & p_{\text{model}}\left(x \mid \theta\right) = p_{\text{latent}}\left(z\right) \left|\frac{\partial G_\theta(z)}{\partial z}\right|^{-1} = p_{\text{latent}}\left(\bar{G}_\theta(x)\right) \left|\frac{\partial \bar{G}_\theta(x)}{\partial x}\right|. \end{aligned}$$

Here, x corresponds to the training space and z to the latent space. The generative model is labeled G_θ, such that $x = G_\theta(z)$. For better readability we use $\bar{G}_\theta = G_\theta^{-1}$ for the inverse of our model.
In the next step we require that the likelihood of our TS, given the optimal model parameters has to be maximal. This is equivalent to saying that our training set is a good representation of all reasonable samples.

$$\begin{aligned} \Rightarrow \quad \mathcal{L}_{\text{INN}} &= -\left\langle \log p_{\text{model}}\left(x|\theta\right)\right\rangle_{p_{\text{data}}\sim TS} \\ &= -\left\langle \log p_{\text{latent}}\ \left(\bar{G}_\theta(x)\right) + \log\left|\frac{\partial \bar{G}_\theta(x)}{\partial x}\right|\right\rangle_{p_{\text{data}}\sim TS}. \end{aligned} \tag{1}$$

The main difficulty at this point is to find a model that is invertible and that allows for a fast evaluation of the Jacobian in Equation 1.
In this thesis we are always using a coupling block architecture to satisfy these two constraints. The general idea is to use simple invertible transformations that are parameterized by a complex neural network. Usually one uses one half of the features, the coordinates of the corresponding samples, to get the parameters of the transformation and the other half is transformed accordingly. Once both halves are transformed, one has to shuffle the input vector to model the correlations of both halves of the vector. This way one can easily invert the full model, as each individual part of it is invertible. A visualization of a coupling block using affine transformations (cf section 3.3.5, "Affine coupling blocks") can be seen in Figure 5.

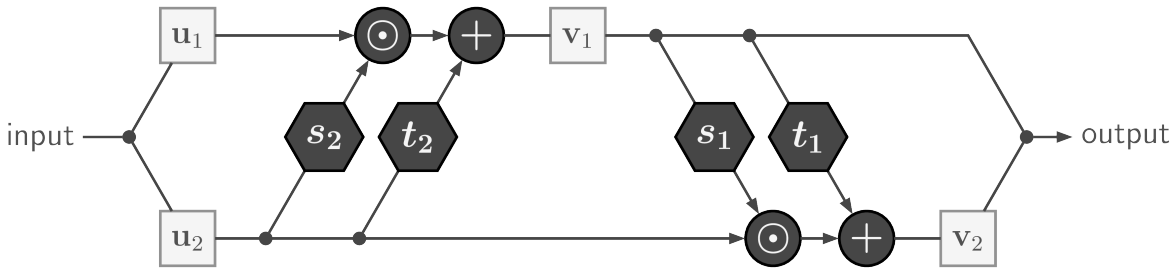

Figure 5: A schematic visualization of an affine coupling block. Taken from [61].

Affine coupling blocks An intuitive, yet not very expressive solution, is to use "affine coupling layers" as coupling blocks for the model:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_1 \odot e^{s_2(z_2)} + t_2\left(z_2\right) \\ z_2 \odot e^{s_1(x_1)} + t_1\left(x_1\right) \end{pmatrix} \Leftrightarrow \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \left(x_1 - t_2\left(z_2\right)\right) \odot e^{-s_2(z_2)} \\ \left(x_2 - t_1\left(x_1\right)\right) \odot e^{-s_1(x_1)} \end{pmatrix}$$

Here x_1 denotes the first half of the input vector and x_2 the second half of it, i.e. $x = (x_1, x_2)$. Equivalently, we use $z = (z_1, z_2)$. The parameters $s_1,\ s_2,\ t_1$ & t_2 are predicted by an arbitrary neural network.
x_2 does not depend on z_1, therefore the Jacobian matrix is an upper triangular matrix. Because of this, its determinant can be easily computed as the product of the diagonal entries in $\frac{\partial G_\theta(z)}{\partial z}$:

$$\begin{aligned} \frac{\partial G_\theta(z)}{\partial z} &= \begin{pmatrix} \partial x_1/\partial z_1 & \partial x_1/\partial z_2 \\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2 \end{pmatrix} = \begin{pmatrix} \operatorname{diag}\left(e^{s_2(z_2)}\right) & \text{finite} \\ 0 & \operatorname{diag}\left(e^{s_1(x_1)}\right) \end{pmatrix} \\ \Rightarrow \quad \left|\frac{\partial G_\theta(z)}{\partial z}\right| &= \operatorname{diag}\left(e^{s_2(z_2)}\right) \operatorname{diag}\left(e^{s_1(x_1)}\right). \end{aligned}$$

Cubic spline coupling blocks A possibility to increase the flexibility of the coupling blocks is to use a more complex invertible transformation. One possibility are monotonic "splines" [62]. The idea is to state some points, called "knots", and functional values and derivatives of the spline at these knots. Then one can build an unique spline as long as one chooses a type of functions that connects these knots. The knots, values and derivatives can be predicted by a neural network, the functional form itself is an important hyperparameter. As all spline transformations are element wise, we receive, again, an upper triangular matrix for the Jacobian and thus an easy and fast evaluation of its determinant. One disadvantage of the spline is that one needs to state an absolute maximum and an absolute minimum of the data range — one needs to know the first and the last knot. There are two solutions to relax this problem. The first is to use a sigmoid preprocessing to map the full real axis onto $[0, 1]$. The other option is to use linear trails, as suggested in [63]. These are just identity functions that are used if the values of the input are smaller than the first knot or larger than the last one.
In this thesis two types of splines are used. One of them is the "cubic spline" as introduced in [64]. The cubic spline coupling block uses cubic polynomials to connect the knots, while ensuring that the full spline is continuously differentiable everywhere, also at the knots. The sub-networks, that are predicting the parameters of the splines f_i, have to predict the positions z_i^K of the knot K and the spline value at this knot $x_i^K = f_i(z_i^K)$. The corresponding derivatives of the spline $\delta_i^K = f_i'(z_i^K)$ cannot be stated and are estimated such that the total spline is monotonous. Here, i corresponds to the feature index of z_1 or z_2, respectively, as also the splines act element wise. We must ensure that $x_i^K < x_i^{K+1}$ and $z_i^K < z_i^{K+1} \forall K \in \{0, ..., \#\text{Knots} - 1\}$ to ensure monotonicity for the knots. This can be achieved by predicting the difference to the previous value instead of the actual value.
Given these prediction we can parameterize the cubic polynomial f_i^K between two knots K and $K + 1$, using the "method of Steffen" [65]:

$$
\begin{aligned}
\xi_i &= z_i - z_i^K \\
w_i^K &= z_i^{K+1} - z_i^K \\
s_i^K &= \frac{x_i^{K+1} - x_i^K}{w_i^K} \\
a_{i,0}^K &= x_i^K \\
a_{i,1}^K &= \delta_i^K \\
a_{i,2}^K &= \frac{3 \cdot s_i^K - 2 \cdot \delta_i^K - \delta_i^{K+1}}{w_i^K} \\
a_{i,3}^K &= \frac{\delta_i^K + \delta_i^{K+1} - 2 \cdot s_i^K}{\left(w_i^K\right)^2} \\
f_i^K(\xi_i) &= a_{i,0}^K + a_{i,1}^K \cdot \xi_i + a_{i,2}^K \cdot \xi_i^2 + a_{i,3}^K \cdot \xi_i^3
\end{aligned}
$$

One can easily check that this parameterization satisfies the given boundary conditions by plugging $\xi_i = 0$ or $\xi_i = w_i^K$ into f_i^K and $(f_i^K)'$, respectively. The only thing that is left, is to ensure that $(f_i^K)' > 0$, everywhere, to ensure global monotonicity. Since we are using cubic polynomials, it is not possible to simply predict the derivatives by a neural network. Even

if we enforce all knot-derivatives to be positive, the spline might have negative derivatives in general because of the possibility of 2 extrema. To prevent this problem, Steffen suggests to use the unique quadratic function that passes through the knots $K-1$, K and $K+1$ to estimate a reasonable derivative from the slopes s_i and the widths w_i [65]. He ends up with the estimate

$$p_i^K = \frac{s_i^{K-1} \cdot w_i^K + s_i^K \cdot w_i^{K-1}}{w_i^{K-1} + w_i^K}.$$

If the quadratic polynomial is not monotonic, one has to correct this estimate, such that one receives the following final estimate for the derivative:

$$\delta_i^K = \begin{cases} 2 \cdot \min(s_i^{K-1}, s_i^K), & \text{if } p_i^K > 2 \cdot \min(s_i^{K-1}, s_i^K) \\ p_i^K, & \text{otherwise} \end{cases}$$

The computation of the inverse and the derivative for a cubic function is possible using existing closed form solutions. However, attention must be paid to numerical stability.
For our implementation of the splines we adapted the code from https://github.com/bayesiains/nflows/blob/master/nflows/transforms/splines/cubic.py.

Rational quadratic spline coupling blocks The downside not being able to actually state the derivatives can be circumvented by using a different interpolation function. In [63] the authors are suggesting to use [66]. In this paper, a parameterization of a monotonic rational quadratic spline (RQS) is introduced, which is able to handle arbitrary (positive) derivatives at the knots. However, to our best knowledge, they do not say anything about uniqueness. Thus, the parameterization is over-restrictive:

$$\begin{aligned} \xi_i &= \frac{z_i - z_i^K}{z_i^{K+1} - z_i^K} \\ w_i^K &= z_i^{K+1} - z_i^K \\ s_i^K &= \frac{x_i^{K+1} - x_i^K}{w_i^K} \\ a_i^K &= \delta_i^K + \delta_i^{K+1} - 2 \cdot s_i^K \\ f_i^K(\xi_i) = \frac{P_i^K(\xi_i)}{Q_i^K(\xi_i)} &= x_i^K + \frac{\left(x_i^{K+1} - x_i^K\right) \cdot \left[s_i^K \xi^2 + \delta_i^K \cdot \xi \cdot (1-\xi)\right]}{s_i^K + a_i^K \cdot \xi \cdot (1-\xi)} \end{aligned}$$

To invert the RQS the authors of [63] show that one has to solve a quadratic equation:

$$z_i = f_i^K(\xi_i) = \frac{P_i^K(\xi_i)}{Q_i^K(\xi_i)} \Leftrightarrow 0 = P_i^K(\xi_i) - z_i \cdot Q_i^K(\xi_i) \equiv R_i^K(\xi_i). \tag{2}$$

As P and Q are quadratic functions, R is quadratic as well. So, the only information that we need to invert the RQS, is which of the two solutions of Equation 2 we should use. This can be found out by realizing that $R(u) = 0$ if $u = \xi(z_i)$

$$
\begin{aligned}
\Rightarrow 0 &= \frac{\mathrm{d}R_i^K}{\mathrm{d}\xi_i} \\
&= \frac{\partial R_i^K}{\partial \xi_i} + \frac{\partial R_i^K}{\partial z_i}\frac{\partial z_i}{\partial \xi_i} \\
&= \frac{\partial R_i^K}{\partial \xi_i} - \underbrace{Q_i^K(\xi_i)}_{>0}\underbrace{\frac{\partial z_i}{\partial x_i}}_{>0} .
\end{aligned}
$$

One can deduce that $\frac{\partial R_i^K}{\partial \xi_i} > 0$, which corresponds to the "+" solution of the quadratic equation (Equation 2).

Conditional INNs As it is the goal of this thesis to generate calorimeter simulations, we want the possibility to generate showers for different incident energies. Therefore, we could either train several INNs, one for each incident energy slice, or try to make the INN learn a conditional distribution. In our case, the first approach is not useful as the discretization of the incident energies corresponds to a loss of information. In the paper [67] a simple possibility is introduced to create a conditional INN. If one passes the conditions to the coupling blocks and concatenates them to their input, the network will use the additional information and learn the corresponding conditional distribution properly.

Other flow types Apart from the coupling block architecture, there are also other types of INNs. They all have in common that they consist of invertible transformations with tractable determinant. Since they are not use in this thesis we are just stating some commonly encountered variations.

1. Masked autoregressive models. These models try to improve their expressivity by making the coupling blocks take more than only half of the features. They are fast in one direction, but other direction's speed scales with the dimensionality of the data [58, 68, 69].

2. Residual Flows. Normalizing flows of the type $G_\theta(z) = z + H_\theta(z)$. The idea is to improve the stability of the training, but problems with the efficient Jacobian computation or the closed form inversion arise [58, 70, 71].

3. Infinitesimal flows are a somehow an infinitely deep version of a Residual flow. They are described by a differential equation $\frac{d}{dt}\mathbf{z}(t) = G(\mathbf{z}(t), \theta(t))$. Here, t represents a "continuous layer index". More information can be found in [58].

Final Comments Even though, INNs are able to model data very accurately, they have two major downsides. The first one was already mentioned. Namely, INNs can theoretically only model distributions that have the same topology as the base distribution. This is limiting the expressivity of the INN a lot in theory. In practice, however, there is a simple solution to this problem. One can add uniform noise to the input data in every batch during training. This way, the data manifold gets smeared out and possible holes are filled by small noise.

If one chooses the noise such that it is numerically smaller than the needed precision of the data, the problem of the topology can be mostly circumvented.
The second problem is a technical problem. The INN scales linearly with the number of bins and the input dimensionality. This makes the INN to be very parameter-hungry in practice. For example the variational auto encoder, as introduced in subsubsection 3.3.7, scales only with the input dimensionality. This results in the INN needing $O(10)$ to $O(100)$ times more parameters than some other network architectures. Limiting the very good model accuracy to rather small input sizes. Solution to this scaling problem of the INN are the major part of this master's thesis.
For the implementation of the INNs we use the python library FrEIA [72].

3.3.6 Bayesian Invertible Neural Networks

If one wants to predict the uncertainties of a network as well, one can use an approach that treats the model parameters probabilistic. Architectures arising from this approach are usually called "Bayesian" networks since one treats the weight distributions via Bayesian statistics. More precisely, one tries to approximate the unknown optimal weight distribution $P(\theta|TS)$, as encoded in the training set, by another distribution $q(\theta)$ using variational inference.
To compare the distributions $q(\theta)$ and $P(\theta|TS)$, we are using the KL-divergence:

$$
\begin{aligned}
\mathcal{L}_{\text{BNN}} &= D_{\text{KL}}[q(\theta), p(\theta|\text{TS}) \\
&= \int d\theta\; q(\theta) \log \frac{q(\theta)}{p(\theta|\text{TS})} \\
&\overset{\substack{\text{Bayes}\\ \text{law}}}{=} \int d\theta\; q(\theta) \log \frac{q(\theta)p(\text{TS})}{p(\theta)p(\text{TS}|\theta)} \\
&= D_{\text{KL}}[q(\theta), p(\theta)] - \int d\theta\; q(\theta) \log p(\text{TS}|\theta) + \log p(\text{TS}) \int d\theta\; q(\theta) \\
&\overset{\text{iid}}{=} \underbrace{D_{\text{KL}}[q(\theta), p(\theta)]}_{\substack{\text{comparison of prior}\\ \text{and approximation}}} - \sum_{x \in \text{TS}} \underbrace{\int d\theta\; q(\theta) \log p(x|\theta)}_{\substack{\text{Log likelihood of x when}\\ \text{sampling } \theta \text{ according to } q}} + \underbrace{\log p(\text{TS}) \int d\theta\; q(\theta)}_{\text{const}(\theta)} .
\end{aligned}
$$

In the last line we assumed that our samples x are independent and identically distributed random variables (iid), enabling us to write the log likelihood over the whole set as a sum over its components. Since BNNs are trained in batches, the KL-loss term needs a prefactor of $w_{kl} = \frac{\text{Batch Size}}{|TS|}$ to correct for the incomplete sum. The structure of this general Bayesian network loss function has a remarkable advantage. It still contains the log likelihood. So, the INN-loss function is just a part of the Bayesian INN loss function. The only difference is that one has to sample the weights in the BINN forward pass according to the learned weight distribution $q(\theta)$ and to add a KL-divergence between the learned distribution and a chosen prior $p(\theta)$. The sampling can be done by replacing the old weights θ by a tuple $(\mu_\theta, \sigma_\theta)$ in each linear layer during the optimization. Then one can use the local "reparameterization trick" [73] to sample the Bayesian weight from a normal distribution of the corresponding mean μ_θ and width σ_θ.

In practice we are usually parameterizing $q(\theta)$ and $p(\theta)$ as normal distributions, as the KL-divergence of two Gaussians has a closed form solution:

$$D_{\mathrm{KL}}\left[q_{\mu,\sigma}(\theta), p_{\mu,\sigma}(\theta)\right] = \frac{\sigma_q^2 - \sigma_p^2 + \left(\mu_q - \mu_p\right)^2}{2\sigma_p^2} + \log\frac{\sigma_p}{\sigma_q}$$

We do not expect to worsen our error estimates this way, as several of this "variational linear layers" are concatenated. Prior work verified this [74, 75].
Putting all of this together we receive the final INN loss function:

$$\begin{aligned}\mathcal{L}_{\mathrm{BINN}} = w_{kl}&\left(\frac{\sigma_q^2 - \sigma_p^2 + \left(\mu_q - \mu_p\right)^2}{2\sigma_p^2} + \log\frac{\sigma_p}{\sigma_q}\right)\\ &- \left\langle \log p_{\mathrm{latent}}\ \left(\bar{G}_\theta(x)\right) + \log\left|\frac{\partial \bar{G}_\theta(x)}{\partial x}\right|\right\rangle_{x\sim p_{\mathrm{data}}, \theta\sim q}.\end{aligned}$$

The first part is going to be called "KL-loss" and the second part is called "INN-loss" when we distinguish between them.
For our implementation we are choosing $\mu_p = 0$ and use σ_p as a hyperparameter. To evaluate the uncertainties of an observable one has to produce n samples and compute the observable for each of the n samples. Then one can compute the mean and the standard deviation over the n different results for the observable.

3.3.7 Variational autoencoders

The INN is not the only possibility to construct a generative model. An alternative solution is to assume the existence of a non-bijective true conditional probability distribution $D(z|x)$, a probabilistic mapping from the input space to a (usually) Gaussian latent space. The resulting model is called variational autoencoder (VAE) [76].
Then, we could train a neural network to assimilate this true distribution by minimizing the KL-divergence between the true distribution and the predicted "encoder" distribution $E(z|x)$:

$$\begin{aligned}\mathcal{L}_{\mathrm{ELBO}} &= D_{\mathrm{KL}}\left[E\left(z|x\right), D\left(z|x\right)\right]\\ &= \sum_{x\in TS}\left\langle \log\frac{E\left(z|x\right)}{D\left(z|x\right)}\right\rangle_{E(z|x)}\\ &= \sum_{x\in TS}\left\langle \log E\left(z|x\right) - \log\left(D\left(x|z\right)\frac{p_{\mathrm{latent}}(z)}{p_{\mathrm{data}}(x)}\right)\right\rangle_{E(z|x)}\\ &= \sum_{x\in TS}\left\langle \log E\left(z|x\right) - \log D\left(x|z\right) - \log p_{\mathrm{latent}}(z)\right\rangle_{E(z|x)} + \mathrm{const}\left(x\right)\\ &\widehat{=} -\sum_{x\in TS}\left\langle \log D\left(x|z\right)\right\rangle_{E(z|x)} + D_{\mathrm{KL}}\left[E\left(z|x\right), p_{\mathrm{latent}}\left(z\right)\right]\\ &= -\sum_{x\in TS}\left\langle \log D\left(x|z\right)\right\rangle_{E(z|x)} + \sum_{x\in TS}\left(1 + \log(\sigma_E(x)^2) - \mu_E(x)^2 - \sigma_E(x)^2\right)\end{aligned} \tag{3}$$

Since we do not know the true posterior distribution $D(z|x)$, we are learning the corresponding likelihood $D(x|z)$, the "decoder distribution" by a neural network as well, using Bayes'

law. For the latent distribution we choose a normal distribution with zero mean and unit variance. For the encoder distribution we assumed a Gaussian as well

$$E(z|x) \propto \frac{1}{\sigma} \exp\left(-\frac{(z-\mu_E(x))^2}{2\sigma_E(x)^2}\right).$$

For the decoder we can assume a different probabilistic models to make the loss function explicit. In the original paper the Gaussian and the Bernoulli distribution are suggested [76]. The advantage of this setup is that we are learning two distributions, instead of using an analytical inverse. This way we can handle different topologies and even different dimensionalities of the latent space and the input space. This enables the VAE to double as a compression tool, as well.
If one wants to generate new samples one can sample from the latent distribution p_{latent} and use the decoder distribution $D(x|z)$ to transform it into a sample in the input space.

Gaussian decoder The normal distribution is usually the first assumption for the decoder distribution. However, the variance σ^2 is typically chosen as a constant and thus not optimized during the training

$$D(x|z) \propto \exp\left(-\frac{(x-\mu_D(z))^2}{2\sigma_D^2}\right).$$

Then the log-likelihood term in Equation 3 becomes a simple mean squared error (MSE)

$$\langle \log D(x|z) \rangle_{E(z|x)} = \frac{1}{2\sigma_D^2} \sum_{z \sim E(z|x)} (x-\mu_D(z))^2 + \text{const}(\sigma_D).$$

Since σ is a free, but constant parameter, it is common practice to treat it as a hyperparameter that has to be chosen by hand. In the literature one often finds $\beta = 2 \cdot \sigma_D^2$. This parameter allows the user to decide whether it is more important that the output looks like the input (small β) or if it is more important that the latent space is close to the prior (large β). The VAEs that are tuning this parameter β are also called β-VAEs [77].

$$\Rightarrow \mathcal{L}_{\text{GVAE}} = \sum_{x \in TS} \left[\sum_{z \sim E(z|x)} (x-\mu_D(z))^2 + \beta \cdot \left[1 + \log\left(\sigma_E(x)^2\right) - \mu_E(x)^2 - \sigma_E(x)^2\right] \right]$$

In practice, the assumption of an elementwise Gaussian likelihood $D(x|z)$ is very restrictive. Since it is usually not fulfilled, one often sees blurry results. In order to prevent this issue, it is common practice to return the mean of the decoder distribution during evaluation, instead of sampling from it. So, usually $\hat{x} \equiv x_{\text{reco}} = \mu_D(z)$ is used. In this thesis this is always the case, when a Gaussian VAE is used.

Bernoulli decoder An alternative to the Gaussian VAE is the Bernoulli VAE:

$$D(x|z) = \lambda(z)^x (1-\lambda(z))^{1-x}.$$

For this assumption one has to replace the mean squared error with a binary cross entropy.

$$\langle \log D\left(x|z\right)\rangle_{E(z|x)} = \frac{1}{\beta} \sum_{z\sim E(z|x)} \left[x \log\left(\lambda(z)\right) + (1-x)\log\left(1-\lambda(z)\right)\right].$$

We decided to keep the β-parameter from the Gaussian VAE for the Bernoulli VAE as well. It does not have a probabilistic justification like before but it is still a possibility to control the importance of the KL term and the likelihood term.
As λ is also the mean of the Bernoulli distribution, it is possible to use $\hat{x} = \lambda(z)$ instead of $\hat{x} \sim D(x|z)$, if the problem at hand allows for a continuous reconstruction. In fact, we are going to use $\hat{x} = \lambda(z)$ whenever we are discussing a (discrete) Bernoulli VAE, just like it is the case for the Gaussian version.
Putting all of this together one ends up with the Bernoulli VAE loss function

$$\begin{aligned}\mathcal{L}_{\text{BVAE}} = &\sum_{x\in TS} \sum_{z\sim E(z|x)} \left[x \log\left(\lambda(z)\right) + (1-x)\log\left(1-\lambda(z)\right)\right] \\ &+ \beta \cdot \sum_{x\in TS} \left[1 + \log\left(\sigma_E(x)^2\right) - \mu_E(x)^2 - \sigma_E(x)^2\right].\end{aligned}$$

Continuous Bernoulli decoder In [78] the so-called continuous Bernoulli distribution is introduced. The idea is to sample a continuous random variable from a properly normalized Bernoulli distribution

$$D(x|z) = C(\lambda(z)) \cdot \lambda(z)^x (1-\lambda(z))^{1-x}.$$

The normalization constant is chosen such that $\int_0^1 \mathrm{d}x \;\; D(x|z) = 1$. Loaiza-Ganem and Cunningham show that it has the following form:

$$C(\lambda) = \begin{cases} \frac{2\cdot \operatorname{artanh}\left(1-2\lambda\right)}{1-2\lambda} & \text{if } \lambda \neq 0.5 \\ 2 & \text{otherwise} \end{cases}$$

This setup also modifies the expectation value of the probability distribution.

$$\langle x \rangle = \begin{cases} \frac{\lambda}{2\lambda-1} + \frac{1}{2\operatorname{artanh}(1-2\lambda)} & \text{if } \lambda \neq 0.5 \\ 0.5 & \text{otherwise} \end{cases}$$

This implies that one should not simply use $\hat{x} = \lambda$ but $\hat{x} = \langle x \rangle$, resulting in a final shift of the reconstructed sample.

Conditioning Also for the VAE we need the possibility to model a conditional distribution

$$D(x|z, c)$$

to sample showers corresponding to different incident energies. Here, we can use the same approach that we used for the INN. We can simply concatenate the condition — or an embedding of it — to the training data and to the latent space. This way, the encoder and the

decoder learn the correlation between the sample and its condition. If one wants to generate the conditional samples, one simply has to append the corresponding condition to the sample from the latent distribution [79].

3.3.8 Further ML techniques

During this thesis some further common ML techniques are used. These are general "tricks" that do not correspond to any of the previous architectures, but to model optimization in general.

Overfitting It is possible that the optimization algorithm is reducing the loss function on the training set (TS), but not on an independent "validation set" (VS). This process is called "overfitting" and is generally undesirable. It means that the model is memorizing the training data instead of interpolating between it and results in a worse model performance than one might expect from the training loss value. To check for overfitting it is a standard procedure to always check the loss value corresponding to a VS during the network training. There are many ways to prevent overfitting. Three of the most popular are "dropout", "batch-norm" and "weight decay".

Dropout Dropout, as introduced in [80], is a technique, where a random subset of the network parameters is set to zero. The subset is resampled every time the forward pass of the network is used. The corresponding hyperparameter is the percentage of the weights that is muted. Usually values between $p = 0.1$ and $p = 0.5$ are used. The effect of the dropout regularization is that the weight-to-weight correlations are weakened. This empirically helps a network to generalize better.

Batch-norm Batch norm is a different method to counteract overfitting. It also helps to train networks with more layers as it stabilizes the gradients and increases the numerical stability [81].
The idea is to compute a running mean μ and a running standard deviation σ over the batches after every layer. Then, one adds a batch-norm layer at this position that normalizes the data on a per feature level:

$$x \leftarrow \alpha \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta.$$

The parameters α and β are learnable, while the parameter ϵ is a small regularization. The downside of batch norm is that the network training becomes more dependent on the batch size. Furthermore, more learnable parameters are added, that slow down the general training procedure.

Weight decay Our third method to counteract weight decay is a regularization of our parameters. If the network is memorizing data, this is usually related to not enough data or to too much flexibility in our parameters. Meaning, one obvious solution to overfitting is reducing the number of network parameters. Weight decay achieves this by adding a L2-regularization to the loss:

$$\mathcal{L} \leftarrow \mathcal{L} + d \sum_{\theta_i} (\theta_i)^2.$$

Due to this regularization, the network will always try to keep its weights as small as possible, which means that it will try to not use too many weights. Here, d is the strength of our weight decay.

Preprocessing and postprocessing Even though, a NN can approximate arbitrary functions, it is empirically better at learning values of $O(1)$. Furthermore, it is easier to approximate sums than products using a NN. Therefore, it is common practice to use a handcrafted preprocessing of the TS before applying the NN to it. After the model evaluation one uses a corresponding postprocessing to remove these modifications.
Typical preprocessing steps are logarithmic transformations as they transform products into sums. Furthermore, a standardization of the data to zero mean and unit variance is usually helpful.

3.4 ML for detector simulations

3.4.1 CaloChallenge

In this section, the domains of detector physics and machine learning are combined to explain the idea behind the CaloChallenge more precisely.
As mentioned before, the goal is to speed up the detector simulations without reducing their accuracy. Therefore, generative machine learning models are trained on data generated by GEANT4. The first task is to find a representation of the detector measurement that our generative model can predict. Several options are possible. The most intuitive possibility is to encode the "hits", the interactions of the particles with the detector, as a point cloud. Meaning that the model is trained to predict an unordered list of coordinates. However, as the number of predictions is not fixed, the application of ML is not straightforward, so this encoding was not used for the challenge. Nevertheless, this is an encoding that will be tested in the future.
The idea that was chosen by the challenge was to interpret the calorimeter measurement as a 3D-image, consisting of voxels. This idea is motivated by the cell and layer structure of the used calorimeters. Thus the combination of the 2D cell index and the layer index result in a three dimensional grid. In this case the number of energy predictions per shower (=voxels) is constant and only their value varies.
However, there is no reason to stick with the low resolution of the detector cells. Since GEANT4 is predicting the energy deposition on an interaction level, the resolution of the learned grid and thus the number of voxels is arbitrary as long as a mapping from the voxels

to the detector cells exists.

3.4.2 Performance evaluation

Classifier Test To measure the performance of the trained models two different metrics are used. The first one was already introduced in subsubsection 3.3.4, the classifier test [39,55,56]. In order to perform the classifier test we were training a classifier to distinguish the GEANT showers from the ML-generated showers. Our main observables in this context were the distributions of the classifier weights and the area under the ROC curve. We started investigating the weight distributions in section 4.3.2, "Classifier weights". Before that we investigated only the AUC values. The use classifier was always consisting of 2 hidden linear layers and 512 neurons per layer. Initially, we used LReLU activation functions with the default slope of $s = 0.01$, trained for 150 epochs with an LR of $2 \cdot 10^{-4}$ and used the model with the best validation accuracy for the final evaluation. For the final weight histograms of dataset 2 and 3 we optimized this classifier by adding dropout with $p_{dropout} = 30\%$ and a LR schedule that reduces the LR by 0.1 if the validation loss was not improving for 10 epochs.

High level observables Our second measure was a collection of high level observables. They are a collection of observables as introduced in [30, 31, 39, 41]. In this section we state the full list of all used observable, even though not all observables were always used:

- The total energy
$$E_{tot} = \sum_{l,i,j} x_{l,i,j}$$
- The total energy E divided by the incident energy E_{inc}
- The total energy deposited in each calorimeter layer
$$E_l = \sum_{i,j} x_{l,i,j}$$
- Quotients between the layer energy depositions of different layers
$$E_{l_1,l_2} = \frac{E_{l_1}}{E_{l_2}}$$
- The center of the shower in each layer along the η or the ϕ coordinate
$$\langle \eta \rangle_l = \frac{\sum_{i,j} \eta(i) \cdot x_{l,i,j}}{E_l} \qquad \langle \phi \rangle_l = \frac{\sum_{i,j} \phi(j) \cdot x_{l,i,j}}{E_l}$$
- Differences between the centroid plots of different layers
$$\langle \eta \rangle_{l_1,l_2} = \langle \eta \rangle_{l_1} - \langle \eta \rangle_{l_2} \qquad \langle \phi \rangle_{l_1,l_2} = \langle \phi \rangle_{l_1} - \langle \phi \rangle_{l_2}$$

- The width of the shower in each layer along the η or the ϕ coordinate

$$\langle \eta^2 \rangle_l = \frac{\sum_{i,j} \eta(i)^2 \cdot x_{l,i,j}}{E_l} \qquad \langle \phi^2 \rangle_l = \frac{\sum_{i,j} \phi(j)^2 \cdot x_{l,i,j}}{E_l}$$

- Differences between the widths plots of different layers

$$\langle \eta^2 \rangle_{l_1,l_2} = \langle \eta^2 \rangle_{l_1} - \langle \eta^2 \rangle_{l_2} \qquad \langle \phi^2 \rangle_{l_1,l_2} = \langle \phi^2 \rangle_{l_1} - \langle \phi^2 \rangle_{l_2}$$

- The energy deposited in the N_{th} brightest voxel per layer
- The activity of the individual layer, meaning the fraction of voxels that have an energy above a certain threshold t

$$a_l = \frac{1}{N} \sum_{i,j} \begin{cases} 1, & \text{if } x_{l,i,j} \geq t \\ 0, & \text{otherwise} \end{cases}$$

- The energy distribution over all voxels

Here, we used the variable x to denote the energy deposition in the voxel identified by the three indices l, i and j. l was symbolizing the layer index, i the index along the η axis and j the index along the ϕ axis. The corresponding coordinates are noted as $\eta(i)$ and $\phi(j)$, respectively. The total number of voxels is labeled N. If the histograms of these observable for the GEANT4 samples and the generated samples "appeared similar" by eye, we considered the samples to be good.

3.4.3 Datasets

In this Master's thesis four different datasets are used to train generative models for detector simulations. They are increasing in difficulty and size:

1. CaloGAN dataset (https://data.mendeley.com/datasets/pvn3xc3wy5/1)
2. CaloChallenge dataset 1 (https://zenodo.org/records/8099322)
3. CaloChallenge dataset 2 (https://zenodo.org/records/6366271)
4. CaloChallenge dataset 3 (https://zenodo.org/records/6366324)

CaloGAN dataset This dataset was not part of the CaloChallenge, but it was the dataset that was already used, when I started my thesis. It was introduced in the CaloGAN papers [30, 31]. In fact it is a collection of three datasets that differ in the particle type of the incoming particle. The simulations are done for photons, positrons and pions (π^+). The dataset was used to train the CaloGAN, a generative network that was trying to simulate a simplified version of the ATLAS detector. It describes a sampling calorimeter made out of several layers of absorbers and active material. As absorber lead and as active material liquid argon was used. However, all hits in the absorber and the active material are used, and the

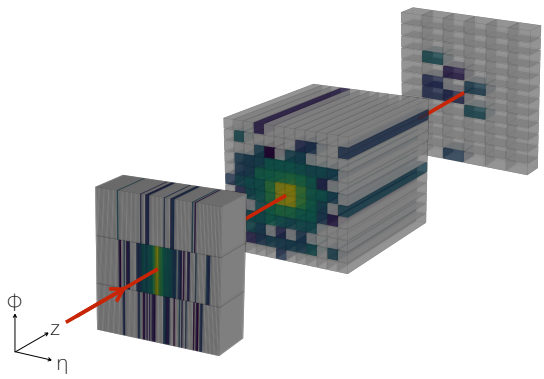

Figure 6: Visualization of the calorimeter setup corresponding to the CaloGAN dataset. Figure was taken from [31].

corresponding energy depositions are summed to end up in only three effective layers. These layers have a thickness of 90 mm, 347 mm and 43 mm, respectively. The full calorimeter has a volume of 480 mm^3. The voxelization of the three effective layers is 3×96, 12×12 and 12×6, respectively. Resulting in a full dimensionality of 504 voxels. The full setup can be seen in Figure 6. For each particle there are 100,000 different calorimeter images in the dataset, corresponding to incident energies uniformly sampled between 1 GeV and 100 GeV. As we were adding noise for our INN setup, we decided to apply a threshold in a non-physical region after our generation process to enable zero-energy voxels. For this threshold we were choosing 0.01 MeV.

CaloChallenge dataset 1 The first CaloChallenge dataset consisted of detector simulations for charged pions and photons. The size was comparable to the CaloGAN dataset, even a bit smaller, but the dataset was more realistic. In the CaloChallenge, also a sampling calorimeter is simulated. However, in this case, only the energies of the active layers are used, making the learning task inherently different to the one of the previous dataset.
The set was used to train the models in the AtlFast3 paper [20] and simulates the ATLAS detector [82]. As photons and pions belong to different calorimeters, the voxelization and the number of layers is different for the two different particle types.
For the set, particles between $\eta = 0.2$ and $\eta = 0.25$ were simulated and local coordinates, orthogonal to the particle trajectory, were introduced (cf. Figure 7, left). For these local coordinates a cylindrical system was chosen, as shown in the right plot of Figure 7. This ensured that our particle is always arriving from the same position and passing through the center of our images.
For the actual voxelization five and seven layers with different voxelization were used for photons and pions, respectively. For photons, the voxelization of the active layers was

$$8 \times 1,\ 16 \times 10,\ 19 \times 10,\ 5 \times 1,\ 5 \times 1$$

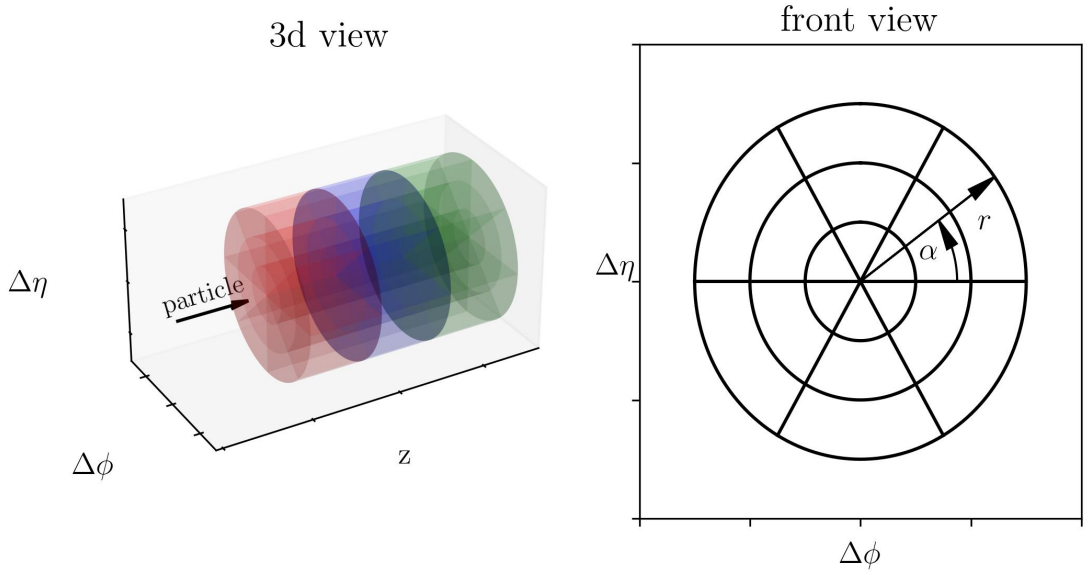

Figure 7: Visualization of the calorimeter setup corresponding to the CaloChallenge dataset 1. Figure was taken from [41].

resulting in 368 voxels. For pions the corresponding voxelization was

$$8 \times 1,\ 10 \times 10,\ 10 \times 10,\ 5 \times 1,\ 15 \times 10,\ 16 \times 10,\ 10 \times 1$$

resulting in 533 voxels. Here, the first index corresponds to the number of angular bins and the second index to the number of radial bins ($n_r \times n_\alpha$).
The incoming particles were simulated at 15 different discrete incident energies between 256 MeV and 4.2 TeV, increasing in powers of two. For the ten lowest incident energies 10000 samples were generated for the five largest incident energies, the following number were used, going from low incident energy to large incident energy:

E_{inc} [TeV]	0.26	0.52	1.04	2.1	4.2
photons:	10000	5000	3000	2000	1000
pions:	9800	5000	3000	2000	1000

For this dataset, the correct value for our final threshold was not really clear. We found significant contributions to the voxel-energy-distribution down to 10^{-2} keV. However, we found that our generation accuracy below keV was not very good and the results became worse, when allowing the network to generate voxel energies lower than that. In the end we chose a threshold of 2 keV.

CaloChallenge dataset 2 The second dataset of the CaloChallenge is conceptually similar to the first dataset. The main differences are that no real world detector was used, that

the incident energies are not discrete and that the number of voxels increased significantly. However, the coordinate system was chosen just like it was done for the first set
For the voxelization, 45 active layers with a binning of 16×9 voxels were used, resulting in 6480 voxels in total. The simulated detector was using a sampling calorimeter with 90 layers, 45 layers of 1.4 mm Tungsten (W) absorbers, each followed by an active silicon (Si) layer of 0.3 mm thickness. The incident energies of the simulated electrons were sampled uniformly between 1 GeV and 1 TeV. Furthermore, the threshold level was given as 15.15 keV.

CaloChallenge dataset 3 For dataset 3 the same setup was used as for dataset 2. Only the number of voxels per layer was increased. Dataset 3 uses a voxelization that ends up with 50×18 voxels per layer, resulting in 40,500 voxels in total.

4 Experiments

4.1 INN

I started my Master's thesis with the work on INNs. More precisely, I started understanding the already working code produced by Luigi Favaro. He was using a conditional INN (section 3.3.5, "Conditional INNs") to describe the CaloGAN dataset. His approach was building upon the CaloFlow papers [39, 40], replacing the masked autoregressive flow with coupling blocks.
The used INN, conditioned on the incident energy of the respective event E_{inc}, was using a coupling block structure with RQS blocks. For each block a fully connected sub-network with three linear layers and 256 neurons, each, was used.
During his experiments Claudius found out that the energies of each layer are much better reconstructed if each layer is normalized to a unit layer energy [39]. In this case the actual layer energies can be learned explicitly and the corresponding predictions can be used to renormalize each layer in a final postprocessing step. In the following the term "normalized energies" refers to the elements of a dataset after this initial normalization step, the term "unnormalized energies" means the elements without this normalization, respectively. The encoding u_i of the layer energies E_i was:

$$u_1 = \frac{E_{tot}}{E_{inc}}, \qquad u_i = \frac{E_i}{\sum_{j=i}^{L} E_j}, \quad i = 1, \dots, L-1. \tag{4}$$

Here, L denotes the total number of calorimeter layers. After the normalization these u_i, in the future called "extra dimensions", were appended to the data for each event.
The usual noise was chosen to be uniform between 0 and $n = 10^{-6}$ and added directly after the normalization, resampled in every batch. The upper limit n will be called "noise width" in the future.
The noise addition was followed by other preprocessing steps. The condition E_{inc} was transformed using a natural logarithm, the (normalized) input together with the extra dimensions was preprocessed using an α-regularized logit function:

$$\text{logit}(k) = \log\left(\frac{k}{1-k}\right), \qquad k = (1-2\alpha)x + \alpha. \tag{5}$$

$$\tag{6}$$

These modifications were needed to prevent calls of logit(0) or logit(1). Several values for α were tried, the best results were achieved with $10^{-10} \leq \alpha \leq 10^{-6}$.
In the logit space the data was standardized using a norm layer that transformed the data to zero mean and unit variance. A schematic overview over the full preprocessing and postprocessing can be seen in Figure 8.
For the training the ADAM optimizer (cf subsubsection 3.3.2) was used, together with a batch size of 256. The learning rate was scheduled for 200 epochs according to an one-cycle LR scheme and confined between 10^{-4} and 10^{-5}.
A numerical summary of the architectural and training details can be seen in Table 1.
Since the hyperparameters and the training in general were already very optimized, there

was not much to do besides understanding the preprocessing and the code structure. I was able to reproduce the results (Figure 9, green line) quickly and moved on to the first real task, adding an uncertainty estimate to the predicted detector simulations. Therefore, I was trying to replace the "normal" INN with a Bayesian version of it (cf. subsubsection 3.3.6).

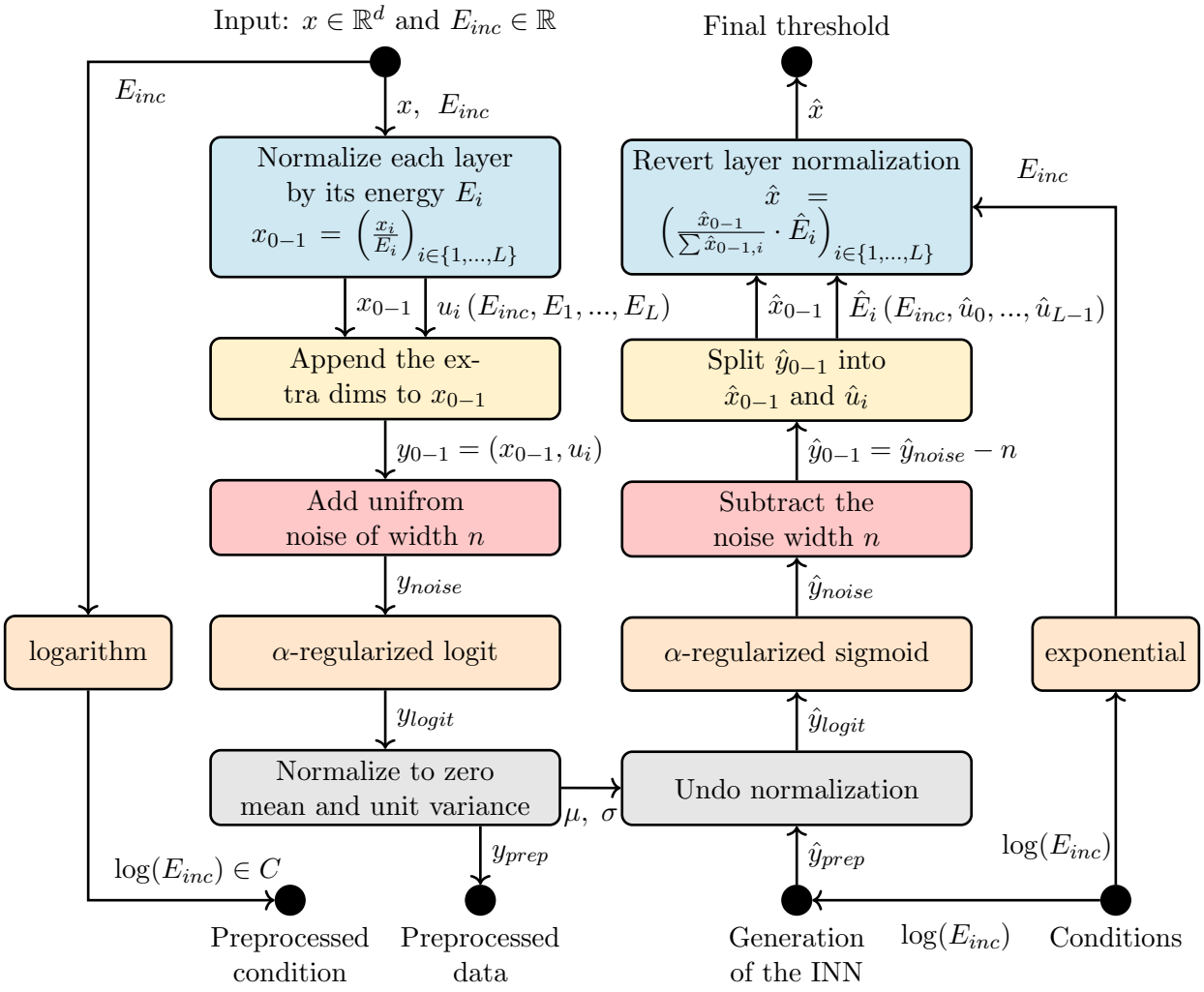

Figure 8: Summary of the INN-preprocessing

4.2 BINN

4.2.1 First approaches

In the beginning I was trying to use the hyperparameters found by Luigi as a baseline and just replace the linear layers in the sub-models by Bayesian layers and the INN loss with the BINN loss. Since the network was doubling its parameter count this way, I decided to start with 400 epochs instead of 200 epochs. However, as shown in Figure 9 (red line), the results were significantly worse.

Parameter	Initial INN/BINN	Smaller BINN	Final BINN	
coupling blocks	RQS	RQS	RQS	
# of layers	3	3	3	
# of hidden neurons	256	32	256	
# of bins	10	10	10	
# of blocks	12	12/18	12	
# of epochs	200/400	400	250	
batch size	256	256	64	
lr scheduler	"one cycle"	"one cycle"	"one cycle"	
max. lr	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	
n	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	
α	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-8}$	
p_{dropout}	5 %	5 %	0 %	
$\log(\sigma_{init}^2)$	-9	-9	-15	} BINN
σ_p^{-2}	1	50	5000	} BINN

Table 1: Network and training parameters of the INN for the CaloGAN dataset.

The first, obvious approach was to variate basically all the hyperparameters in Table 1. But these changes did not really improve the results. The approach to train longer or to use larger learning rates even resulted in unstable training. Usually, between 300 to 600 epochs the INN loss became noisy. It started to diverge if we trained even longer (cf Figure 10). The main problem with this observation was that the KL part of the loss did not converge until then. So, we had to either stop the training before the network was converging, or receive exploding losses.
In the end, this diverging behavior of the Bayesian network was the main problem that I was facing during this part of my master's project.
The first approaches to fix this problem were purely empirical. We thought that it was an architectural problem and thus tried to change the network structure to fix the instabilities. Therefore, we tried the following modifications:

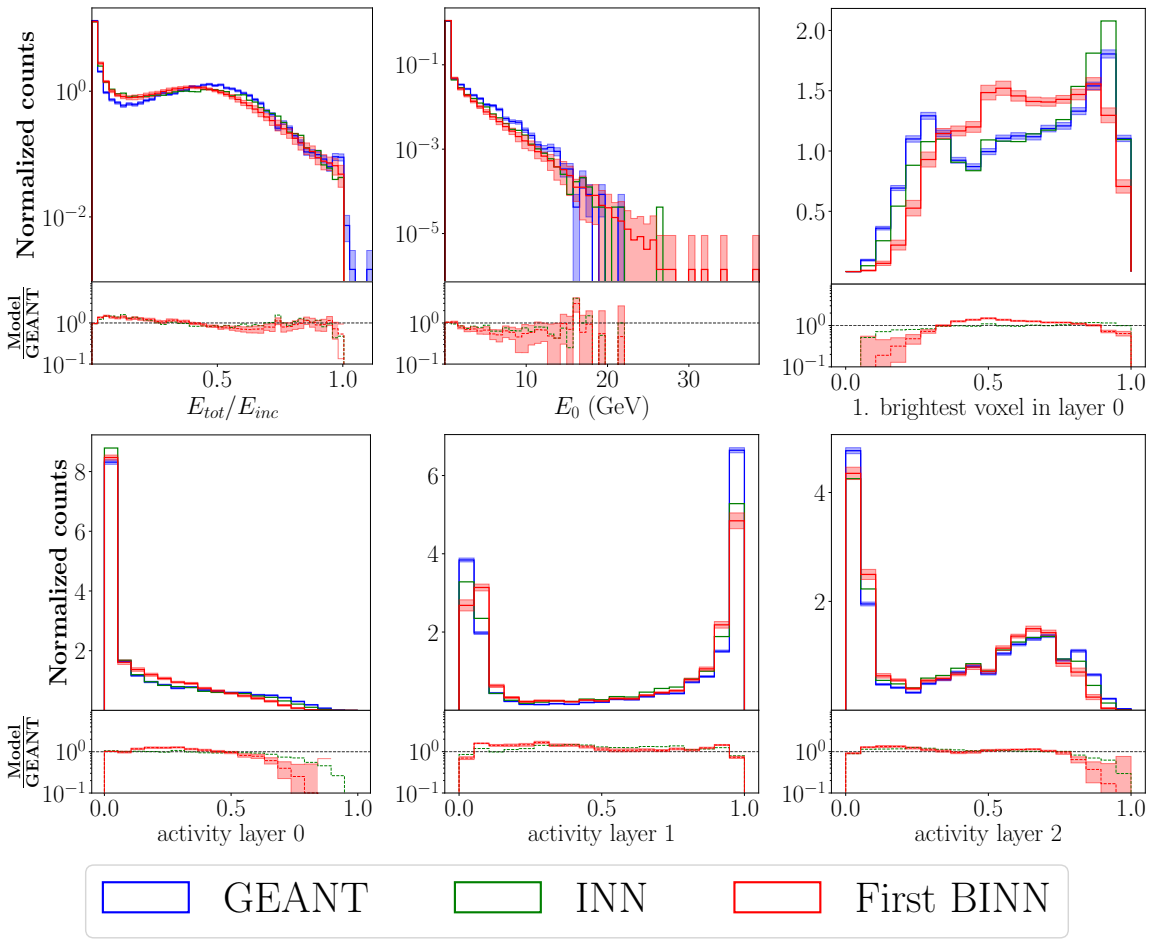

Figure 9: Out-of-the-box results for the pion dataset using the INN and its Bayesian version, trained with Luigis framework.

- Replace the RQS coupling blocks with affine or cubic spline coupling blocks.
- Change the way the noise was added.
- Clip the corresponding gradients.
- Try to change σ_p and σ_{init}.

Different coupling blocks This approach did not help at all. In fact the results with the different coupling blocks were so much worse, that I never tried to move away from RQS splines afterwards. For all tried coupling blocks the training was unstable but for the cubic and especially the affine coupling blocks the results before the instabilities were significantly worse.

Fix the added noise Another idea was to not resample the added noise with each batch but to sample only once for the whole training set. The motivation was that resampling the

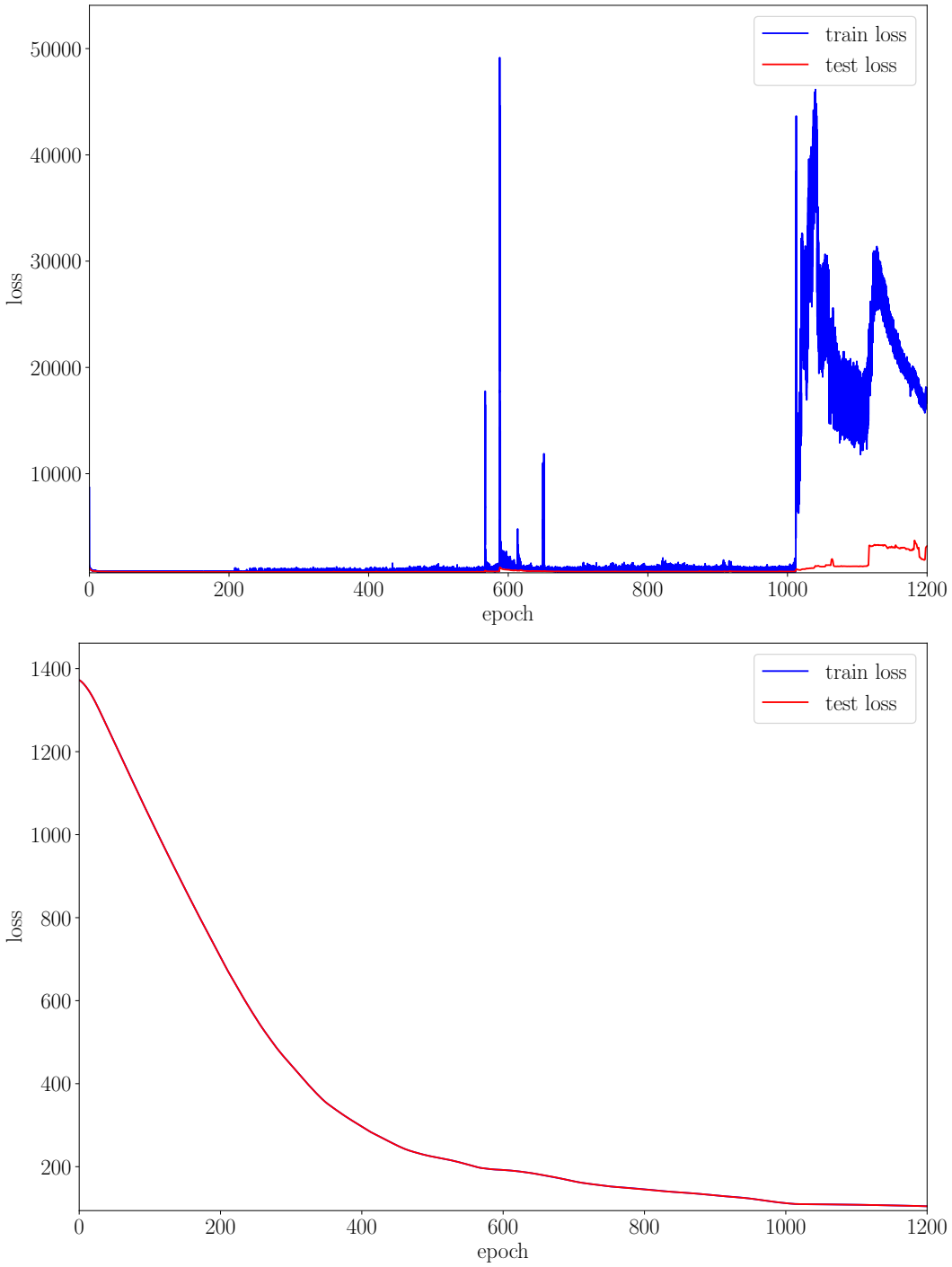

Figure 10: An example of a diverging INN-loss (top) and the corresponding KL-loss (bottom). Around 600 epochs, the loss starts spiking. The training settles down afterwards, but the optimum is worse than before. After 1000 epochs the loss and the gradients diverge. The corresponding KL loss was not converging until then.

noise would artificially increase the size of the training set and mess with the prefactor of the KL loss term w_{kl}, resulting in the different speeds of convergence. However, fixing the noise

did not prevent the instabilities nor did it increase the convergence speed of the KL loss. However, it reduced the quality of our generations. Because of that went back to resampling the noise every batch. In this context we also varied the theoretically determined prefactor of the KL loss term itself. As expected, it made the results only worse.

Clip the gradients At some point we noticed that the diverging loss was always accompanied with a strongly increasing gradient. To exclude the possibility that the large gradient were a reason and not a symptom we started monitoring the maximum of the gradients by default and tried clip the gradients. However, the clipping was not preventing the instabilities, proving that they were just a symptom.

Modify σ_p and σ_{init} The most promising approach during the initial empirical test phase was the modification of the prior width σ_p and the σ_q initialization σ_{init}. Changing the $\log(\sigma_{init}^2)$ mean value from -9 to -25 was enabling a more stable training and results that were almost comparable to the INN results. However, the latter was only true if we decided to stop before our σ_q parameters were able to reach σ_p, a time just determined by the learning rate and the number of epochs as all σ_q parameters were moving towards σ_p. Once the σ_q were of the order of magnitude of σ_p we were still experiencing the diverging behavior. Since we were just creating an enforced deterministic limit of the Bayesian network, we decided to not stay with this pseudo-solution. After all, we had no reason to trust the uncertainty estimates if the σ_q parameters did not converge at all.

4.2.2 Reduce the input dimension systematically

At this point we realized that the diverging behavior might be a more fundamental problem. So, we started to analyze it more systematically by monitoring the σ_q development closely during the training and by reducing the input dimensionality to make the training task easier. To do this we iteratively reduced the input dimensionality. First we reduced the training to just one calorimeter layer, then to 20 hand chosen voxels, and finally we ended up with only the extra dimensions.
During this increasing simplification we were able to confirm that the problematic point was where the σ_q parameters where reaching σ_p. We especially observed that almost all σ_q parameters were moving closer to σ_p — an observation that was not obvious (cf Figure 11). In fact we were expecting that the sigmas would stop at some point, namely when the INN loss term would start to suffer from the intrinsic uncertainty.
Our interpretation for this phenomenon was (and is) that the network had too many (Bayesian) parameters. If the network is too large, the only restriction for the majority of its parameters is the KL loss term. By its structure it is motivating the μ_q to become 0 and the σ_q to be similar to σ_p. It seems that the network is unable to stop this "internal shutdown" when it is affecting its generation quality if it has too many parameters. In other words, at some point the resulting meaningless Gaussian noise is weakening the networks generative power.
This hypothesis is supported by the observation that, after the INN loss term becomes noisy and when $\sigma_q \to \sigma_p$, peaks in the INN loss term appear. During this peaks the network seems

to restructure itself, removing the useless noisy internal pathways from its important neural connections. But the important thing is that after every peak the INN loss gets worse. (cf Figure 10, epoch 600).
Our first approach to solve this problem was to simply reduce the number of parameters in the network until the training became stable. This point was reached when we trained only on the energy dimensions with 12 blocks, but only 32 neurons per block instead of 256 neurons. However, the generation quality suffered from this reduction in parameters a lot. We found that the training was also stable when increasing the number of blocks to 18 without increasing the number of neurons per block. This increase in the number of blocks improved the results for the three energy dimensions that much that we decided to slowly increase the dimensionality again. In Figure 12 we show a comparison between the old setup, with 256 neurons, trained for only 200 epochs due to stability reasons and the new setup with 32 neurons. For 12 blocks the results are worse than the large network generations, for 18 blocks we receive comparable results to before while being stable during training.

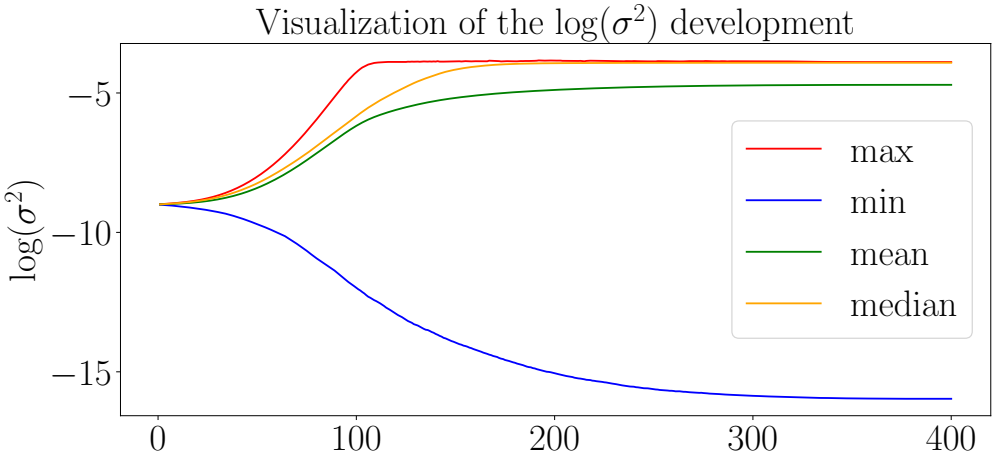

Figure 11: Here we show the development of the widths used to sample our Bayesian parameters. Almost all of the $\log(\sigma_q^2)$-parameters are moving towards the prior width σ_p and more than the half of them are reaching it around epoch 200. For this plot we are using the stable BINN on the three extra dimensions, introduced in subsubsection 4.2.2, as the unstable BINNs were crashing, before the convergence became obvious.

4.2.3 Going back to the full dimensionality

To increase the dimensionality of the learned data in a meaningful and systematic manner we decided to "create" three downscaled datasets from the original CaloGAN dataset. In practice we applied iterative average pooling. We used different pooling kernels for the x-downscaling and the y-downscaling as we had no square-like data. For the x-downscaling we used a $(2, 1)$ kernel with a stride of $(2, 1)$ and for the y-downscaling we were using a $(1, 2)$ kernel and a $(1, 2)$ stride, respectively. These average pooling operations were iteratively applied until the desired resolution was achieved. Simulating a dataset with a lower resolution.

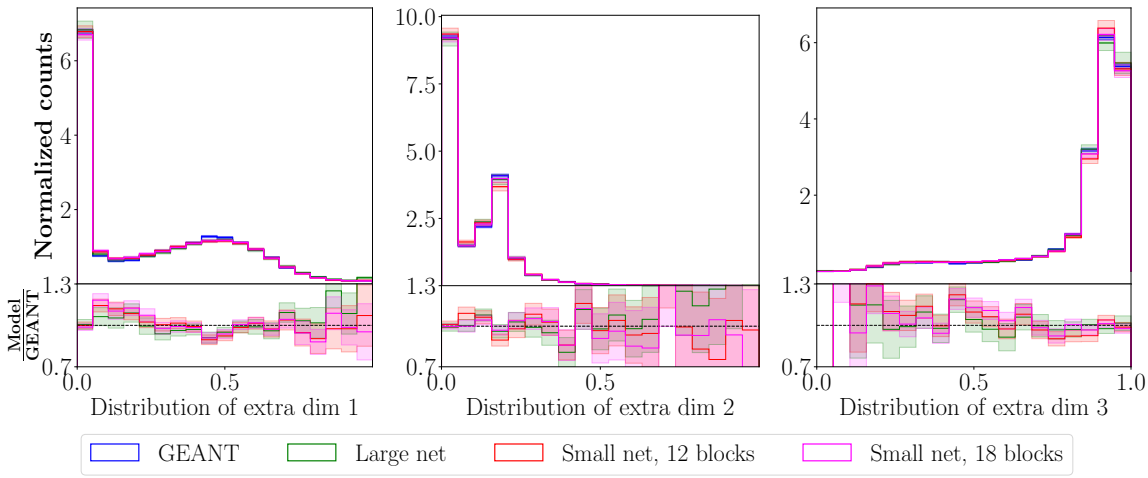

Figure 12: Here we show the effect of the reduced internal size, when training only on the extra dimensions. The green line corresponds to an early stopped Bayesian INN with 256 neurons and 12 blocks. The red line to a stable BINN with 32 neurons and 12 blocks and the purple line to a BINN that is still stable with 32 neurons and 18 blocks. The green line is on average the best, but the purple line is almost as good. The small network with fewer blocks seems to have problems getting some peaks right. For example the main peak around 0.2 in the second extra dim and the maximum close to 1 in the third extra dimensions show a mismatch between its generation and the corresponding ground truth.

This way we generated three additional smaller datasets that we were using to slowly increase our input dimensionality.
During the iterative increase we found that the small number of parameters with more blocks was helping to improve the stability without losing (a lot) expressivity in comparison to the early stopped large Bayesian INN.
However, as the results were also not improving to the quality of the pure INN, we had to keep looking for a possibility to reduce the number of Bayesian parameters without going down in expressivity. At this point we were able to regain our lost expressivity by training longer. However, being able to do both would be the solution to achieve the same generation power that the non-Bayesian INN was showing.

4.2.4 Final solution for the unstable training problem

The final solution, that was actually solving the problem, was to not use Bayesian layers for all layers in the block, but only for the last layer in each block, and normal linear layers for everything else. This means that the sub-networks of the INN, the networks that were predicting the splines' parameters, were sampling their weights only in the end. This is not in a conflict to our derivation of the Bayesian INN (cf subsubsection 3.3.6). In the derived loss function we found that it is necessary to sample the network parameters according to their learned distribution. However, the parameters of the INN are not (directly) the parameters of the sub-networks, but the parameters in the splines. So, as long as the splines' parameters are sampled, the sampling in the Bayesian INN loss function is satisfied. The only possible

downside is a reduced flexibility in the uncertainty description. Therefore, it is necessary to verify that the uncertainties are similar to the old uncertainties and to the predicitions of an ensamble of INNs.
This different approach was enabling us to increase the number of neurons to the amount of the original INN — 256 — without experiencing instabilities. Furthermore, we found that the modifications of σ_p and σ_{init}, described in section 4.2.1, "Modify σ_p and σ_{init}" were applicable here as the uncertainty estimates were indeed trustworthy. In Figure 13 we show that the uncertainties are similar to the predictions of an INN ensemble of ten networks. We see that the results are surprisingly good. Except for the cut in the upper left plot the predictions of the new Bayesian version are better than the pure INN and for this failure mode the uncertainties are raising. This also shows, that the uncertainty predictions work.
We do not show the BINN of the reduced size to keep the figure clear, as it was similar to the large version, shown before. The generations for photons and electrons are shown in Figure 14 and Figure 15, respectively. We did not compare this results to an ensemble to save resources. Since the problem of the unstable Bayesian INN was solved with this new Bayesian structure we decided to use this "simplified Bayesian structure" in the future to generate uncertainty estimates with a BINN and to move on to the next problem — the scaling properties of the INN.

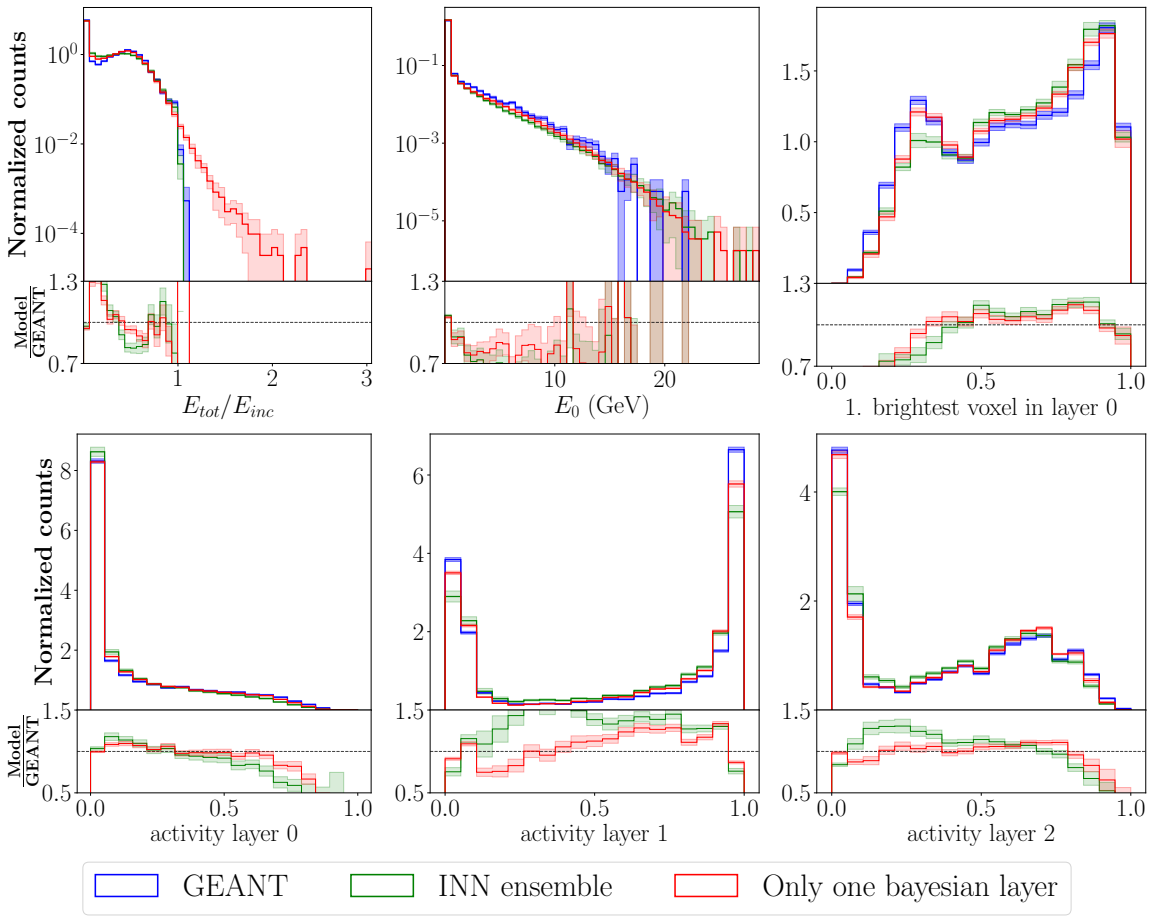

Figure 13: Final BINN results with only one Bayesian layer per coupling block for the pion dataset. We also show the uncertainty predictions from an INN-ensemble to verify the uncertainty estimates.

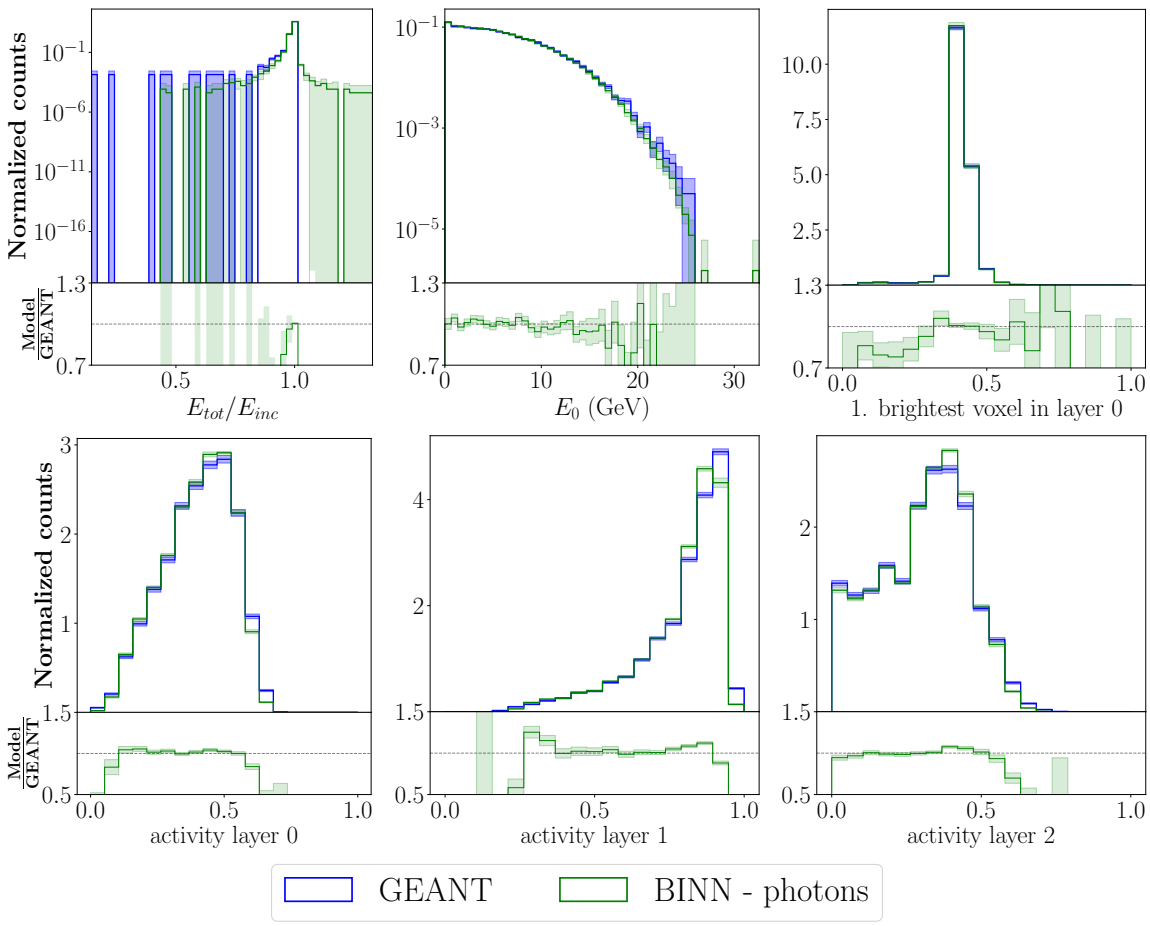

Figure 14: Final BINN results with only one Bayesian layer per coupling block for the photon dataset.

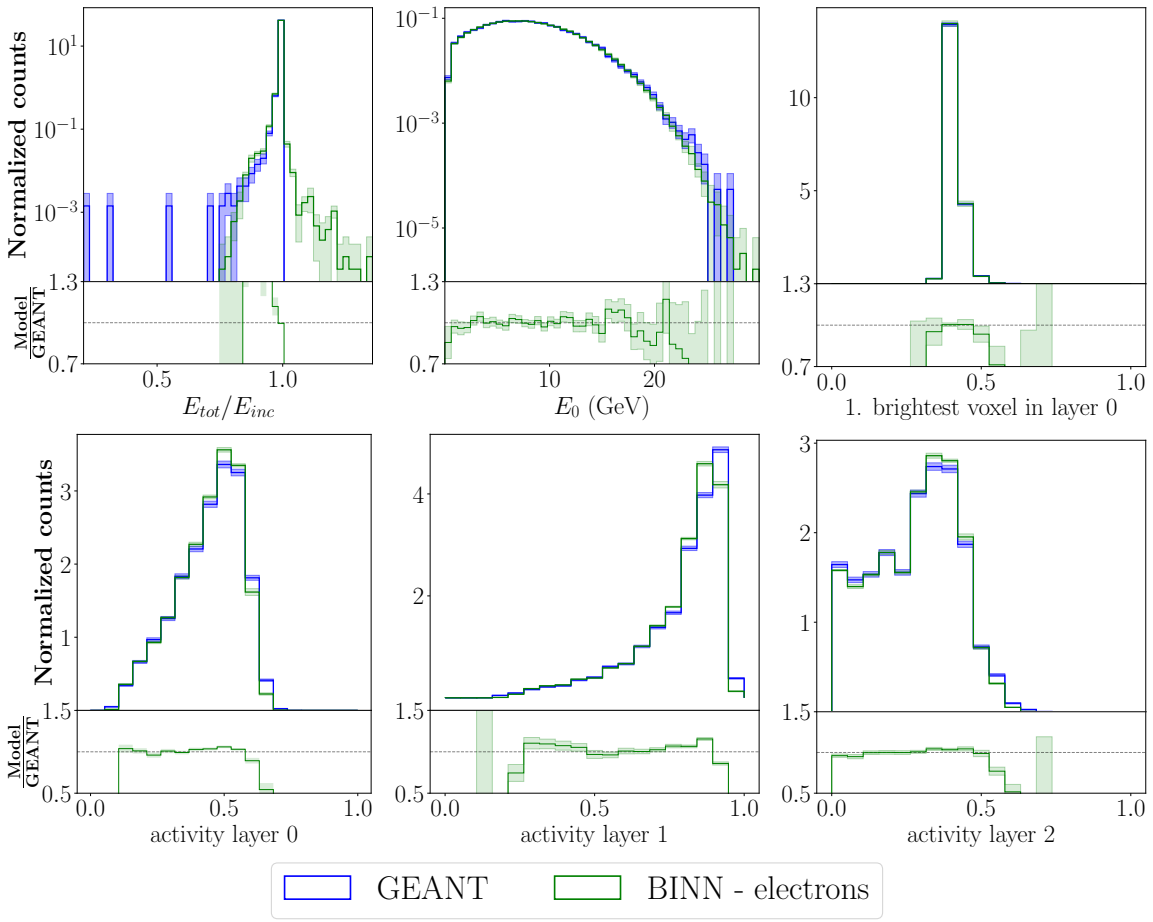

Figure 15: Final BINN results with only one Bayesian layer per coupling block for the electron dataset.

4.3 Gaussian VAE

The next part of my master's project was motivated by the master's thesis of Robin Rombach [83, 84] and the recent paper "CaloMan" [85].
As explained before, INNs have a major drawback, namely their difficult applicability to high dimensional datasets (section 3.3.5, "Final Comments"). The effect is that small amounts of voxels can be generated with almost unrivaled accuracy but large amounts cannot be produced at all without new approaches. A viable solution is to project the data onto a lower-dimensional "latent" space that ideally matches the data manifold's intrinsic dimensionality. Operating in this latent space allows the INN to learn and generate data effectively and accurately, provided that a suitable "upscaling" method can reverse the mapping back to the original space. This approach offers two advantages: reduced computational complexity and the elimination of the need for data noise. My master's thesis primarily focuses on implementing this concept through the use of VAEs.

4.3.1 CaloGAN dataset

We started our VAE approach with a vanilla Gaussian VAE, consisting of an encoder network and a decoder network. The encoder was predicting a mean and a sigma and the latent space was sampled according to those values. For the decoder we found best results by returning the mean predictions, instead of sampling in the output as well. As loss we were using the Gaussian VAE loss with β parameter (section 3.3.7, "Gaussian decoder"). Our MSE loss was comparing the reconstructed and the original showers in the normalized space. We also tried to apply the MSE to the unnormalized space but found the reconstructions to be worse. Furthermore, we continued with the CaloGAN dataset for the beginning. More precisely, we used the pion dataset to optimize our setup and the electron and photon dataset to verify that the architecture changes generalize to other particle types. In fact, the pion results appeared to be easier for the VAE setup, a circumstance that is not completely clear to us, as the pions should be physically more difficult to describe.
As the INN can be trained in the latent space after the VAE training, it is possible to first get good reconstructions from the VAE and then use the INN to generalize the reconstructions to generations. Moreover, the INN was usually not the major problem, so we will mostly focus on the reconstruction quality of the VAE.
We started our experiments with a rather small VAE. Its encoder network consisted of hidden layers with 450, 300 and 150 neurons, respectively, and a latent space dimensionality of 50. The decoder was symmetrically designed with layers containing 150, 300, and 450 neurons. For all of our experiments we were using such a symmetric setup. Because of that we will only mention the encoder's neurons in the future. We did not find any improvements when making the network deeper at this point. However, we did not try to increase the size of the hidden layer above the input dimensionality, as this inflation was intuitively not the way towards a compression — a mistake as we realized later.
The training parameters included a learning rate of $5 \cdot 10^{-4}$, a β-parameter of 10^{-9}, and 500 training epochs. The low β-value prioritized reconstruction quality over the similarity of the latent space to a normal distribution, aligning with the INN's capability to adapt this space to be Gaussian, anyway.
For the preprocessing we started from the INN version and kept the layer normalization,

the extra dimensions and the logit preprocessing. For the logit transformation we used an α-parameter of 10^{-6}. However, we decided to not learn the extra dimensions with the VAE but with the INN. To achieve this we used the extra dimensions as an additional condition for the VAE. Furthermore, we initially omitted noise, as its role in data smearing seemed counterproductive to explicit manifold learning. A schematic overview of our VAE setup and the corresponding initial preprocessing can be seen in Figure 16 and Figure 17, respectively. For the no-noise policy, we were choosing n, the noise width, to be zero. A summary of the initial VAE hyperparameter setup is given in the "Initial VAE parameters" column of Table 2.

Parameter	Initial VAE parameters	Final VAE parameters
# of hidden neurons	450, 300, 150	4500, 1000, 150
# of epochs	500	500
Dim latent space	50	50
batch size	256	256
lr scheduler	Constant LR	Constant LR
lr	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
n	0	$1 \cdot 10^{-6}$
α	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$
β (KL-loss prefactor)	$1 \cdot 10^{-9}$	$1 \cdot 10^{-5}$
γ (Data-loss prefactor)	1	$1 \cdot 10^{4}$
δ (Logit-loss prefactor)	0	1

Table 2: Initial and final network and training parameters of the VAE for the CaloGAN dataset.

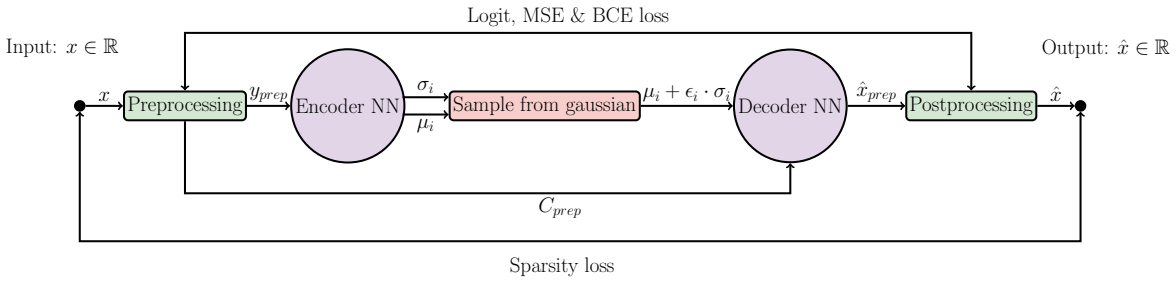

Figure 16: Main pipeline of our VAE setup. Also all losses that are tested in this thesis are shown for future reference.

Bad activity plots and logit loss Our first problem was that the VAE was not learning the activity of the detector layers (cf. green line in Figure 18). We came to the conclusion that this was an effect of the used loss function. The MSE, when applied directly to the normalized data, cannot see a difference between a 10^{-5} GeV and 10^{-4} GeV voxel, if a $O(1)$ normalized voxel energy deviates by $\approx 1\%$, as it is just summing up the absolute differences. This behavior is not what we want as our data spreads over several orders of magnitude. The

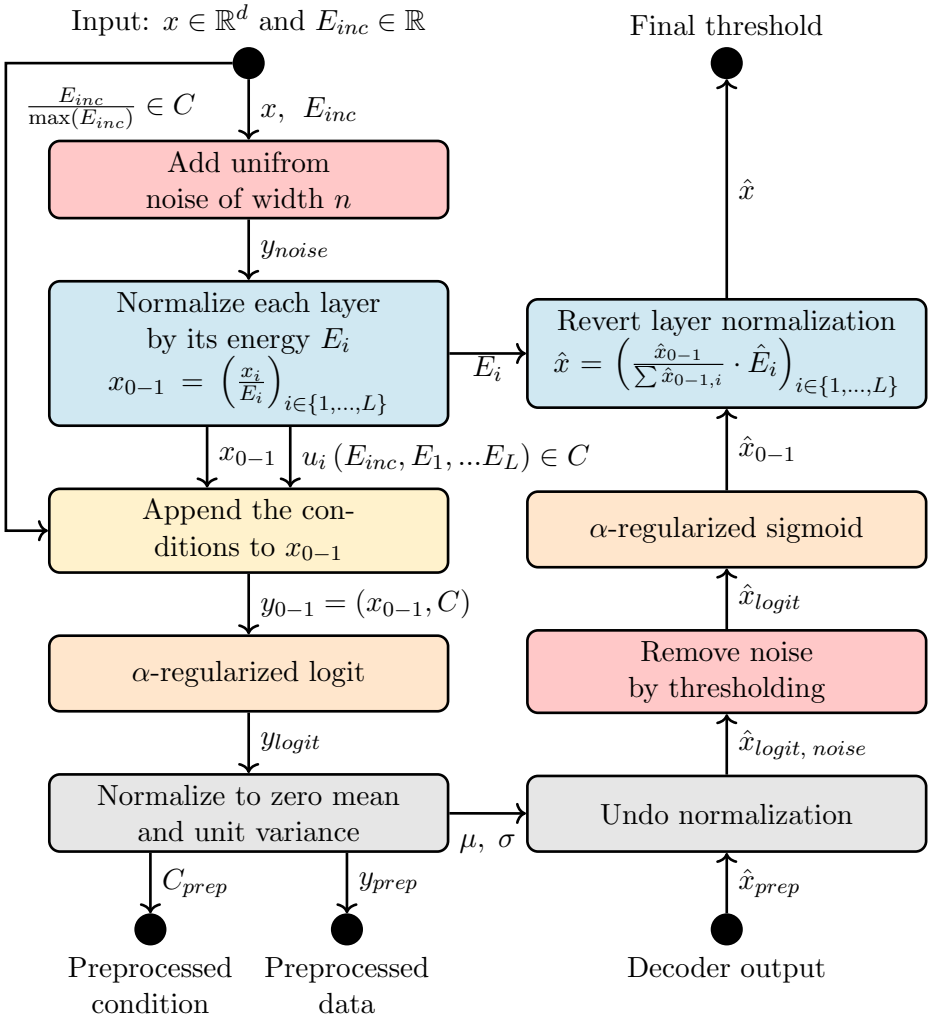

Figure 17: Summary of the first version of the VAE-preprocessing

fact that the activity of our layer is not learned is just a sign that the low energy regions are not reconstructed faithfully.
The most intuitive solution to this problem is to use a logarithmic or logit preprocessing, for the data and the loss. However, when using it on its own, this "logit-MSE" is not able to reconstruct our showers faithfully. Using a MAE instead of a MSE was slightly improving the results, but the reconstruction quality was still not satisfying. Especially the brightest voxels were not learned accurately. This makes sense since the logit is trying to get the orders of magnitude for the data in the normalized space right. This means that the absolute difference for the brightest voxels may differ the most. This, however, is problematic as we enforce the total energy to be correct. So the absolute difference in the brightest voxel can have a negative effect on the other voxels as well, since it might dominate the normalization procedure (cf. Figure 18, red line).
Our best solution was to simply combine the two loss terms, "data-loss" and "logit-loss", weighted by a prefactor that will be called γ in the future. This loss sets a relative boundary for the data in the normalized space and an absolute boundary for the data in the unnor-

malized space. The problematic part is that the constant γ has to be chosen such that the absolute boundary is dominant for the voxels that carry a significant amount of energy. This "loss tinkering" approach was the best solution that we were able to find for the Gaussian VAE (cf. Figure 18, yellow line).
As a result we continued with the following loss function:

$$L_{\text{ELBO}} = \gamma \cdot \text{MSE}\left(x_{0-1}, \hat{x}_{0-1}\right) + \delta \cdot \text{MAE}\left(x_{logit}, \hat{x}_{logit}\right) - \beta \cdot \sum_i \left(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2\right).$$

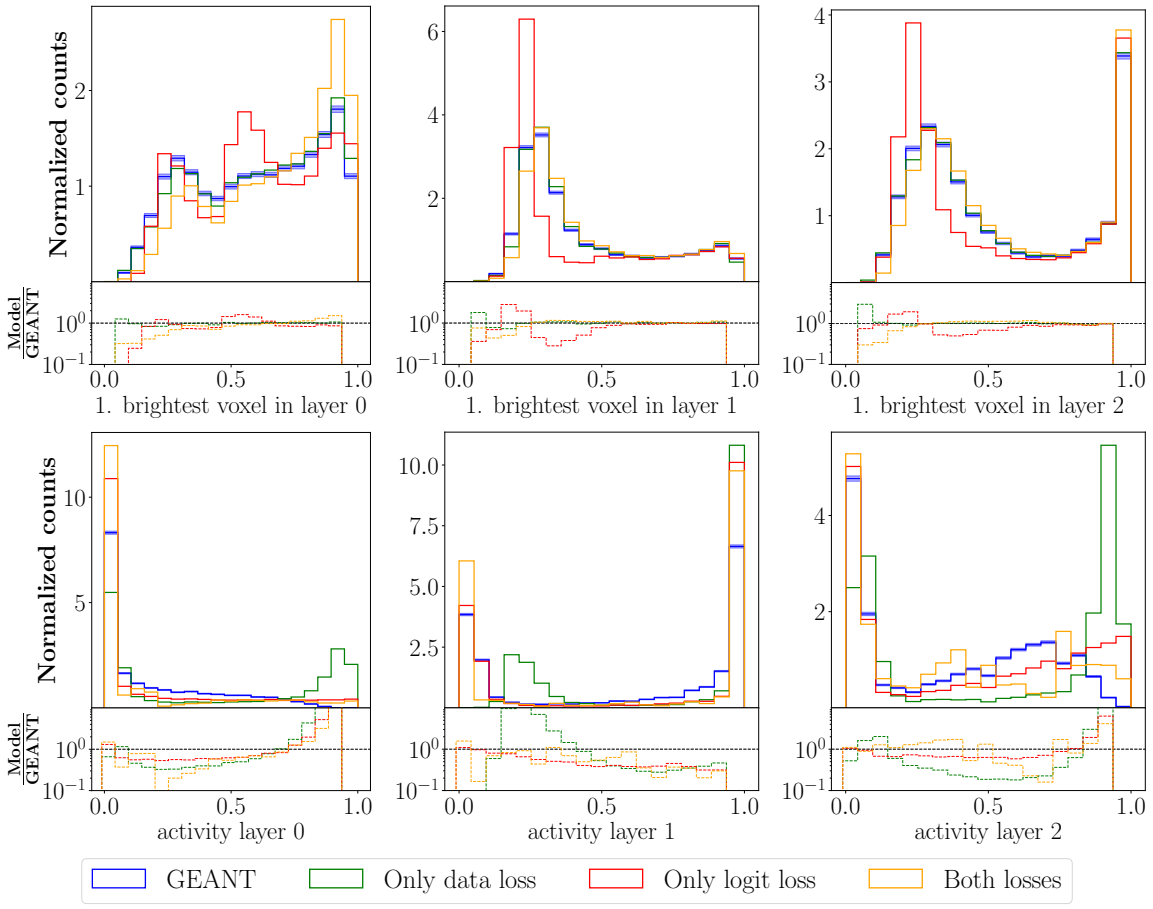

Figure 18: Histograms of the brightest voxels and the activity for three different loss setups. We show the vanilla MSE loss for the unnormalized and the logit space. As a third line we show the best compromise using a combination of a MAE "logit-loss" and a MSE "data-loss". The shower shapes for these approaches were consistently worse than the INN equivalents. We will focus on this problem later, so we do not show them here to save space. We are using a Poissonian square root error estimate for the GEANT data, but no error estimate for the reconstructions. The reconstructions should reproduce the observables per event and should thus not be affected by a statistical uncertainty.

Try noise Even though we expected no improvement from the addition of random noise, we wanted to make sure that it really is not helpful. Surprisingly, we found for the Gaussian VAE that adding a small amount of noise was actually improving the activity and the shower shape reconstructions (cf. Figure 19, red line). However, the effect was strongly dependent on the chosen noise level. In the shown plots we were using a noise level of $n = 10^{-6}$ GeV. The noise was added before the normalization. This way we prevented the noise-strength to depend on the total energy of the shower.
To remove the noise, we first tried to threshold the data after the decoder in the unnormalized space at 10^{-6} GeV, as well. However, this implied the need to first do all the postprocessing and then move back to the logit space for the logit loss. Problematically, we encountered severe numerical instabilities with the evaluations of the type of $\text{logit}\left(\text{sigmoid}\left(\text{logit}\left(x_{0-1}\right)\right)\right)$, that we were not able to resolve at this point. Later, when we were able to fix this instabilities the reconstructions got a lot better (cf. section 4.3.2, "Restructured noise layer").
Our solution at this point was to use the monotonicity of the logit function — if we want to apply a threshold t in the normalized space, we can just pull it into the logit space by applying the logit to it:

$$x_{0-1} < t \Leftrightarrow x_{logit} < \text{logit}\left(t\right).$$

As we want to move our threshold from the unnormalized space to the logit space, we needed a possibility to apply the normalization to the threshold. The problem was that each layer was first normalized and then scaled to match the requested layer energy during unnormalization. This means that the unnormalization effect changes if the data is thresholded, as the sum over the layer is reduced. This means that the thresholding itself changes the normalization, even if the threshold is adjusted properly for each voxel.
To resolve this issue we assumed the effect of the thresholding on the total sum to be small and the resulting self dependency to be negligible. Therefore, we simply divided the threshold by the true layer energies (corrected by the expected noise added per layer) and pulled this per-layer-threshold to the logit space. As we were able to threshold in the logit space this way, we did not need the detour to the unnormalized space for the logit loss anymore.
During our experiments we found the cutoff in the voxel-energy-distribution to be not a straight line, since we ignored the mentioned self dependency. We were able to fix this problem by normalizing to the noisy layer energy instead of the noiseless layer energy. This way we were not able to produce lower layer energies than ≈ 10 MeV in layer 1 and 2 as each of the $O(100)$ voxels was increasing the normalization goal by $O(0.1)$ MeV. However, at this point we were believing in the statement that the layer energies below 10 MeV should be physically less relevant. Interestingly, also the classifier had more trouble distinguishing showers with the modified norm from real showers, than before. Therefore, we were using the modified norm until we change the noise structure in section 4.3.2, "Restructured noise layer". The final threshold was therefore computed as

$$\tilde{t}_l = \text{logit}\left(\frac{n}{E_l + \frac{n}{2} \cdot |L|}\right)$$

$$x_{logit} \leftarrow \begin{cases} x_{logit}, & \text{if} \quad x_{logit} > \tilde{t} \\ \text{logit}(\alpha), & \text{otherwise.} \end{cases}$$

Here, E_l is the incident energy of layer l and L the set of all voxels corresponding to layer l.

One closing remark to the addition of noise in general:
We realized later that for other VAE approaches the noise was not needed at all. So, for the VAE the noise was probably just "hiding" the difficult information of the data, mainly the low energetic voxels. This makes the VAE focus on the high energetic voxels, resulting in data that can be learned better with a MSE.
So, as a conclusion: If the addition of noise helps during the VAE training, something might be wrong with the model and its assumptions. Nevertheless, this is an insight that we made much later, such that we kept using the noise until we discarded the Gaussian VAE model.

Attention and larger network Next, we tried to use a kind of attention mechanism in our VAE to enhance the reconstructions. It was just an exercise for myself as I had to learn about transformers and we did not find any improvement because of the attention. However, we found a large improvement because of the resulting inflation of the size of our first network layer. By increasing our first layer by a factor of ten and our second layer by ≈ 3 we found significantly better histograms for our reconstructions. Mainly the shower shape variables were profiting from the increased flexibility of our encoder and decoder network. The improvement can be seen in Figure 19, comparing the yellow and red line. During our later datasets we observed this behavior frequently and we think that it can be explained the following way:
Even though we expect our physics to be low dimensional, there is no reason why the mapping between the simple latent space and the data space should be "simple" as well. In the end this mapping seems to need a lot of flexibility empirically. Later, we found that this inflation is mainly needed locally, on a per calorimeter layer level. This means, that we do not need a big input layer connecting many detector layers, if we have more than just four detector layers. An important fact as our scaling would have been quadratic otherwise. Meaning, the VAE would at some point be less efficient than just using an INN in the first place.

Add the INN in the latent space At this point we started investigating the INN and the generative power of the model. Therefore we encoded our data once and trained our INN only on the encoded data and the extra dimensions. For the generation we used only the decoder to translate the samples from the latent space to the data space. For the INN we used the hyperparameters that are summarized in Table 3. We did not use a preprocessing for the INN, except a log-preprocessing of the incident energy E_{inc} as everything else was "preprocessed" by the VAE.

For the INN training we tried three different approaches:

1. Train the INN on the actual latent space, gained after sampling according to the encoder μ and σ predictions, once.
2. Train the INN directly on the μ and σ predictions of the encoder network.
3. Train the INN on the actual latent space, but do the sampling step every epoch again.

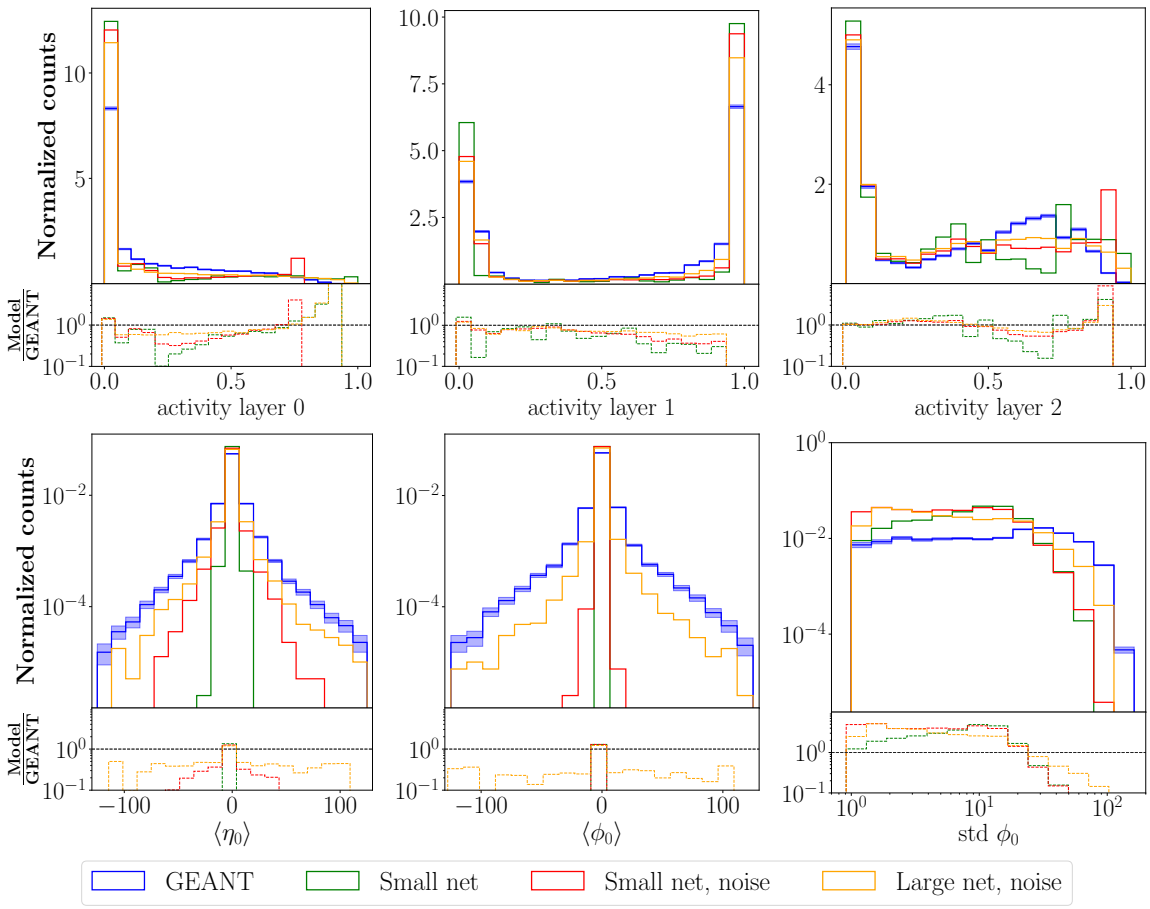

Figure 19: Histograms of the activity and some shower shape variables for the small and large network. For the latter we also show the effect of adding noise. All three nets use the "data-MSE" and the "logit-MAE".
We found no big differences in the brightest voxel plots and a similar behavior across all shower shape plots.

The first approach was not working very well. Our explanation for this problem is that the VAE is storing relevant information in the widths, that are very hard to recover, when sampling only once in the latent space. One obvious solution is to pass the information stored in σ to the INN.
The straightforward method, where we simply add the σ to the INN prediction and sample according to the generated μ and σ values works as well as we were expecting. Using this setup, the generation histograms in are good agreement with the reconstruction histograms. However, this approach has the downside, that the input space of the INN is twice as large, which is not optimal for a compression tool.
The third approach is trying to combine both approaches using the noise setup that was previously used for INNs as well. If one trains the INN on the latent space after sampling once, it will not work well. We have seen this problem already during our analysis of the Bayesian INN (cf. section 4.2.1, "Fix the added noise"). Furthermore, we can interpret the sampling step of the VAE as the addition of some noise. From this point of view it should

improve our INN results if we resample the "noise" in the latent space every epoch. As the encoder network is deterministic, the forward pass has only to be done once, so this approach is not slower than the other two. But, in contrast to our expectations, the result using the third approach was not much better than the first approach and significantly inferior to learning μ and σ.
In the end we kept the second approach as it was working very well. The VAE was always a bigger problem such that we decided to not invest to much time into the specific INN implementation. In Figure 20 we show one sample plot for each of the three approaches. One can clearly see that the INN version that predicts μ and σ is outperforming all other methods by a large margin. Furthermore, we found that for the small latent space size that we were using, the INN scaling never was a problem.

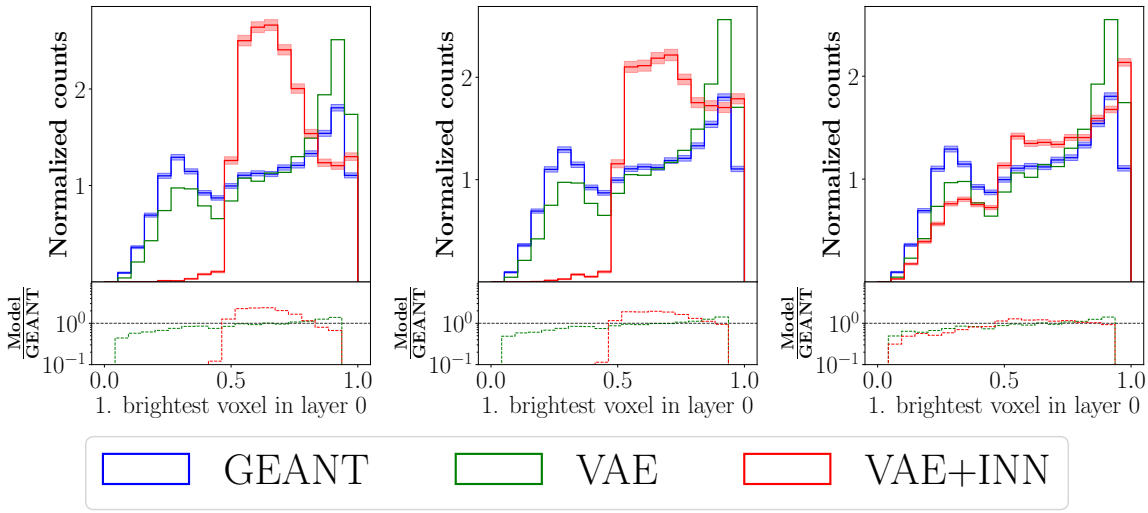

Figure 20: Different approaches to add the INN.
(left) Train INN on latent space after sampling, (mid) train INN on latent space after sampling, resample in every batch, (right) learn the widths and the means with the INN. Here, and in the future, we are adding a Poissonian square root error estimate for the generated data.

High level loss Motivated by [86] we tried to add a "high level" loss to make the VAE explicitly learn specified physical observables. Therefore, we added a MSE comparison of some requested high level observables to our total loss. Specifically, we tried the layer activity and the shower means. The activity did not work as it was not differentiable and thus not contributing any gradient. The centroid loss was also not very helpful. Obviously it was improving the mean plots a lot, but at the cost of making some other plots worse. In other words it was just moving the errors from one plot to another, but not fixing them. Therefore, we did not use any high level losses for the CaloGAN dataset.
Later we tried a different activity loss, that did not use a step function but a steep sigmoid. Using this approach we were able to receive a reasonable gradient. Furthermore, this activity loss was improving all of our histograms. However, it was never able to improve the classifier

Parameter	Internal INN
coupling blocks	RQS
# of layers	3
# of hidden neurons	32
# of bins	10
# of blocks	18
# of epochs	200
batch size	256
lr scheduler	"one cycle"
max. lr	$1 \cdot 10^{-4}$
p_{dropout}	0

Table 3: Network and training parameters of the INN trained in our latent space. As the latent space is low dimensional, we found a small INN to be sufficient.

AUC. It seems that the high level losses introduce artifacts that can be classified easily. More about this activity loss will follow in section 4.4.1, "Activity loss"

Final results At this point we decided to move on to the "interesting" datasets, the datasets that were needed for the CaloChallenge. The main reason was that we had a working VAE framework and mediocre results and we did not see any further obvious improvements. After all, we used the CaloGAN dataset only because we were familiar with it due to our previous studies. The final generated VAE+INN results can be seen in Figure 21 (pions), Figure 22 (photons) and Figure 23 (electrons). One can clearly see, that the electrons and photons showers are harder for the VAE setup. We are not entirely sure, why this is the case, but it might be an artifact from the reduced randomness of the VAE. While the INN receives a random input vector of the same dimensionality as the input, the VAE only receives as many random variables as the size of the latent space. So, it might be not enough randomness in the VAE model to describe electromagnetic showers faithfully. The hadronic showers are more complex and the voxels are probably stronger correlated, the compression might therefore, counter intuitively, be easier. However, as we were not able to improve the generation quality by increasing the latent dimensionality, we were not able to confirm this assumption. We also trained classifiers of the generated showers. The corresponding scores are shown in Table 4. The final hyperparameters are visible in the right column of Table 2.

Particle type	Accuracy	AUC
Positrons	0.9876 ± 0.0007	0.9988 ± 0.0002
Photons	0.9873 ± 0.0011	0.9986 ± 0.0001
Pions	0.8147 ± 0.0012	0.9096 ± 0.0012

Table 4: Classifier test results for the generation using VAE and INN. Is is obvious that the classifier had only "trouble" with the pion dataset. The electron and the photon dataset were not generated well with our used setup. The uncertainties were estimated as the standard deviation over five training runs of the classifier.

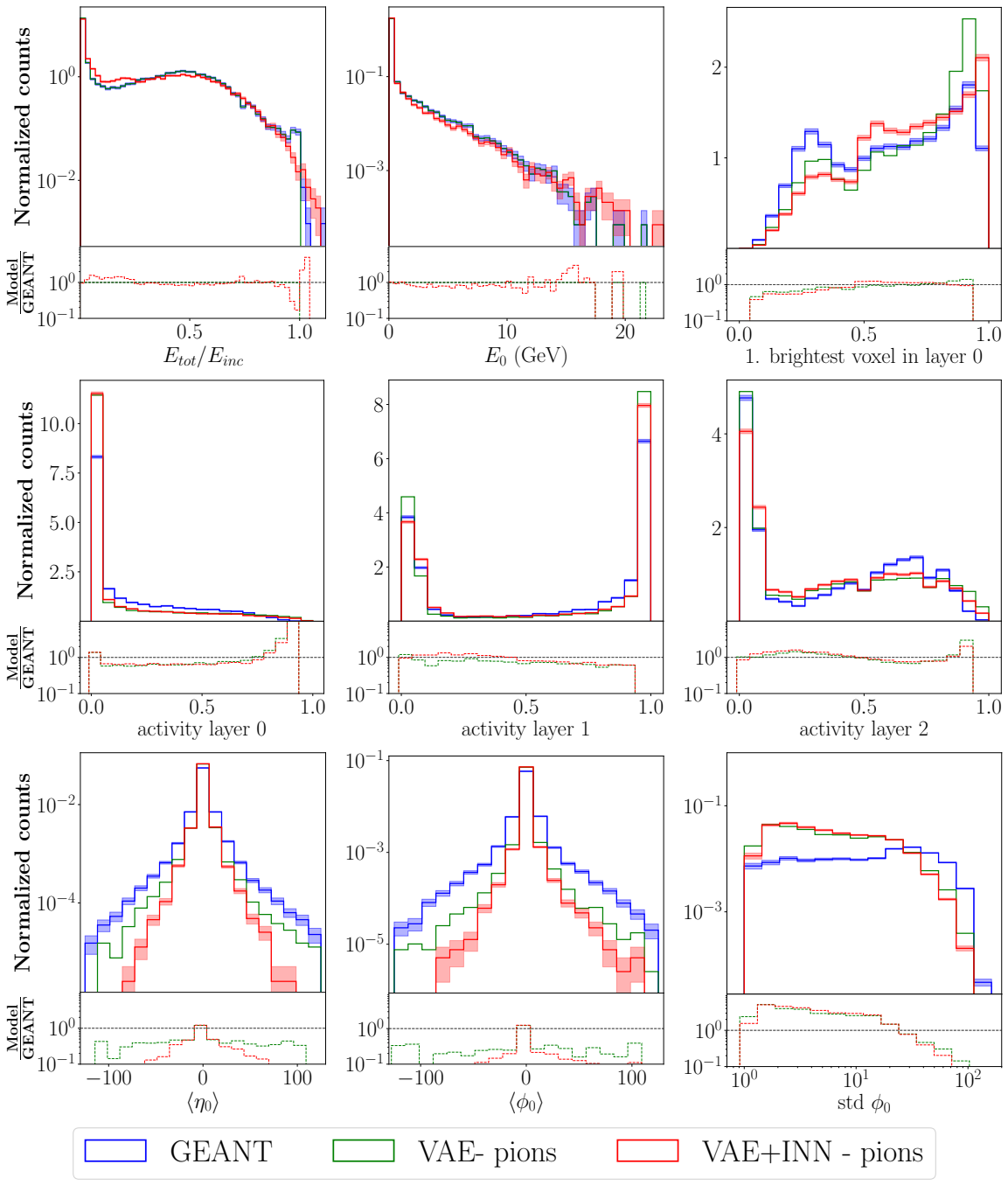

Figure 21: Final results for the CaloGAN pion dataset using the VAE+INN setup.

4.3.2 CaloChallenge dataset 1

Once we started with dataset 1, we decided to focus on the photon set. For the CaloGAN dataset we were expecting the pions to be more difficult, but found that the network had more trouble with the electrons and photons.
With the CaloChallenge dataset we made this observation again. The pion reconstructions were consistently better than the photon reconstructions and improving in a similar manner.

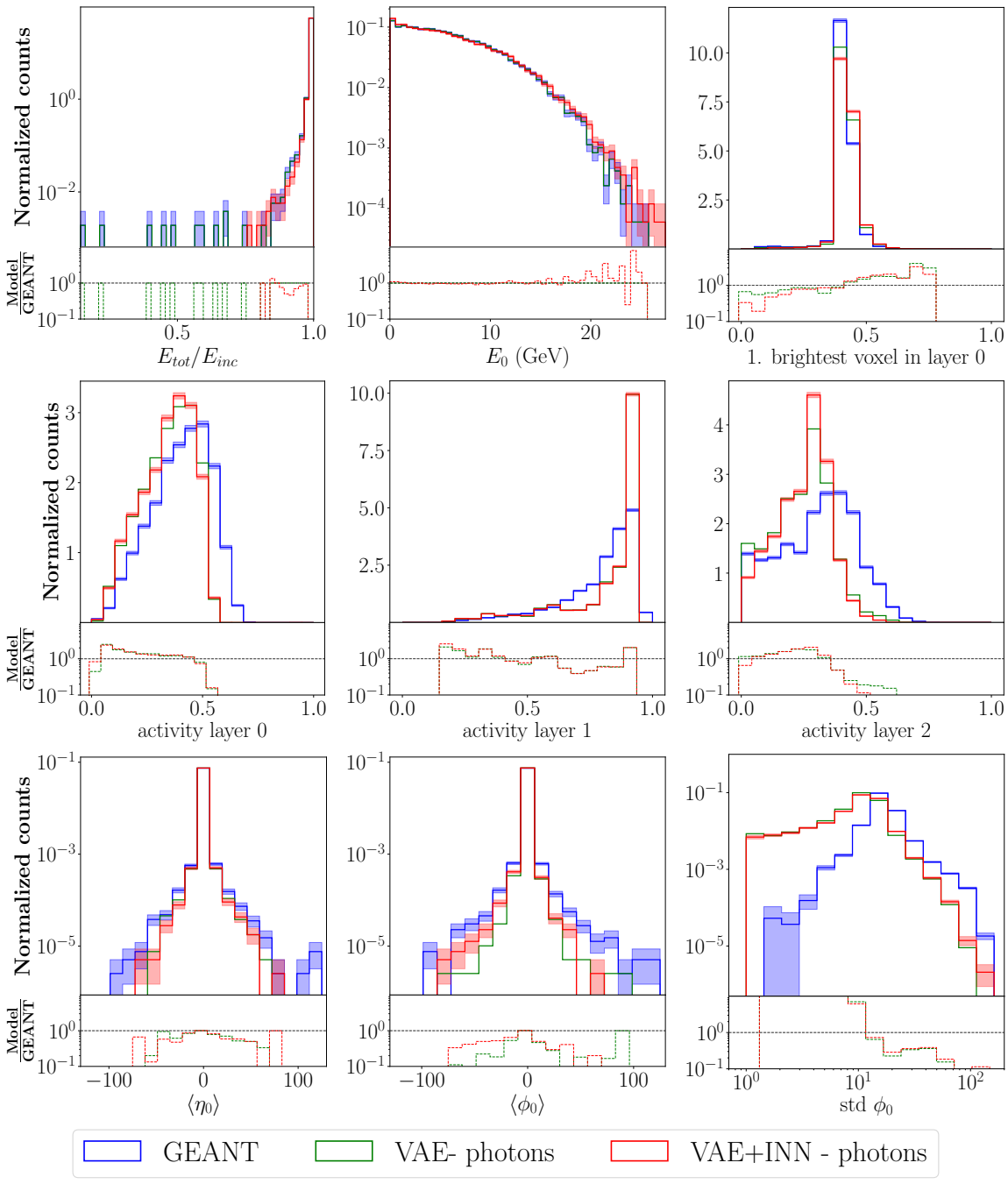

Figure 22: Final results for the CaloGAN photon dataset using the VAE+INN setup.

First Observations When starting with the first dataset of the CaloChallenge, using the same architectural setup as before, our first observation was that our model was overfitting for this dataset. We only scaled our data x by 0.5 with respect to the incidence energy E_{inc}

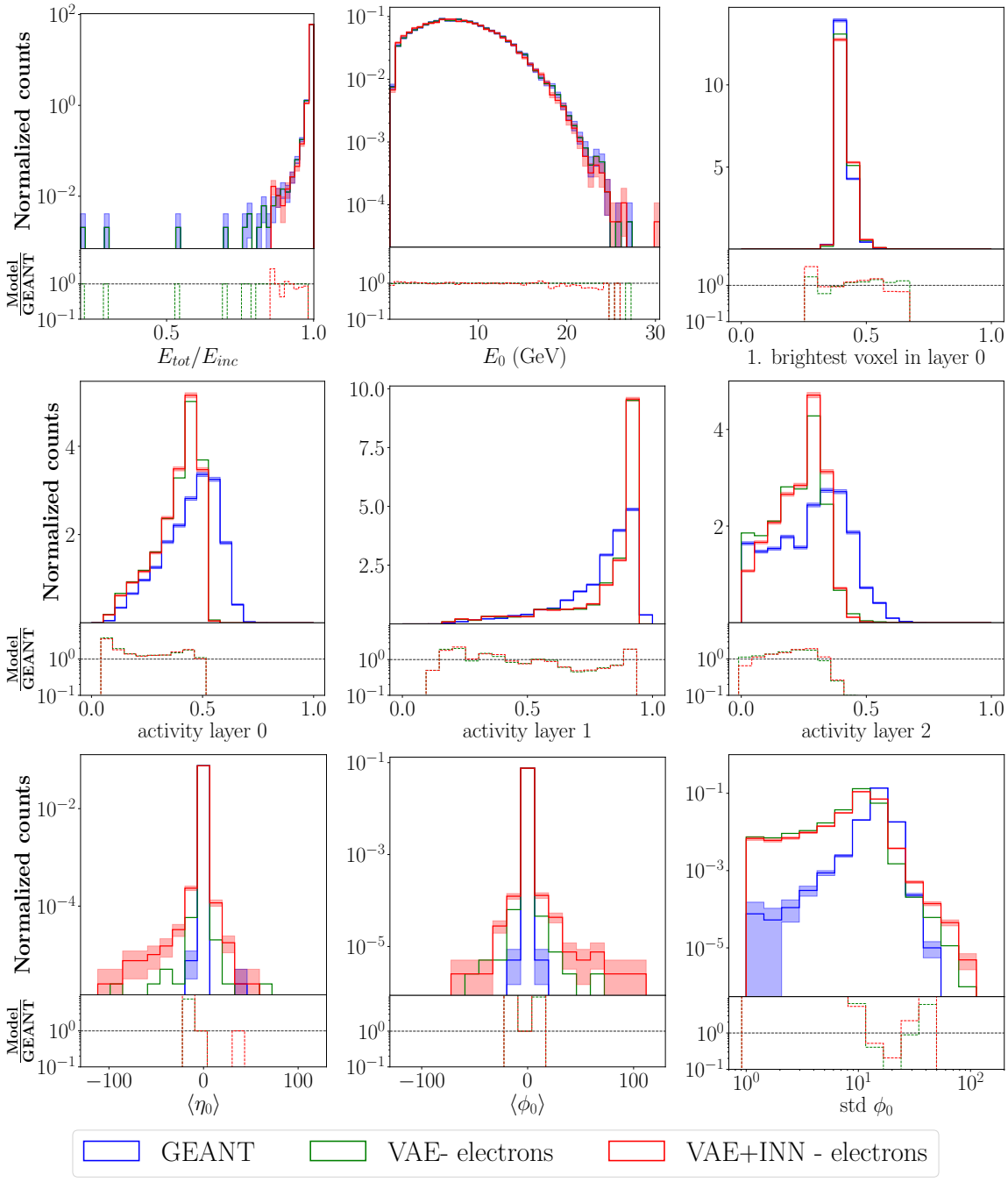

Figure 23: Final results for the CaloGAN electron dataset using the VAE+INN setup.

to ensure

$$\sum x < E_{inc}$$

for almost all of our voxels. This is a necessary step to ensure that our extra dimensions are between zero and one. For the remaining cases, where $\sum x > E_{inc}$ (0.9% pions, 0.01% photons), we clipped our extra dimensions. This factor of $\frac{1}{2}$ also explains the oddly-specific threshold of $t = 2$ that we used for this dataset (cf section 3.4.3, "CaloChallenge dataset 2").

The corresponding model parameter was not fine-tuned, but a simple one.
The validation loss started separating from the training loss, when we were increasing the network size in section 4.3.1, "Attention and larger network". However, up to this point, the validation loss was not increasing, just falling slower than the training loss. Since the network is "truely" overfitting for this dataset (Figure 24), we investigated the reason for this generalization deficit.

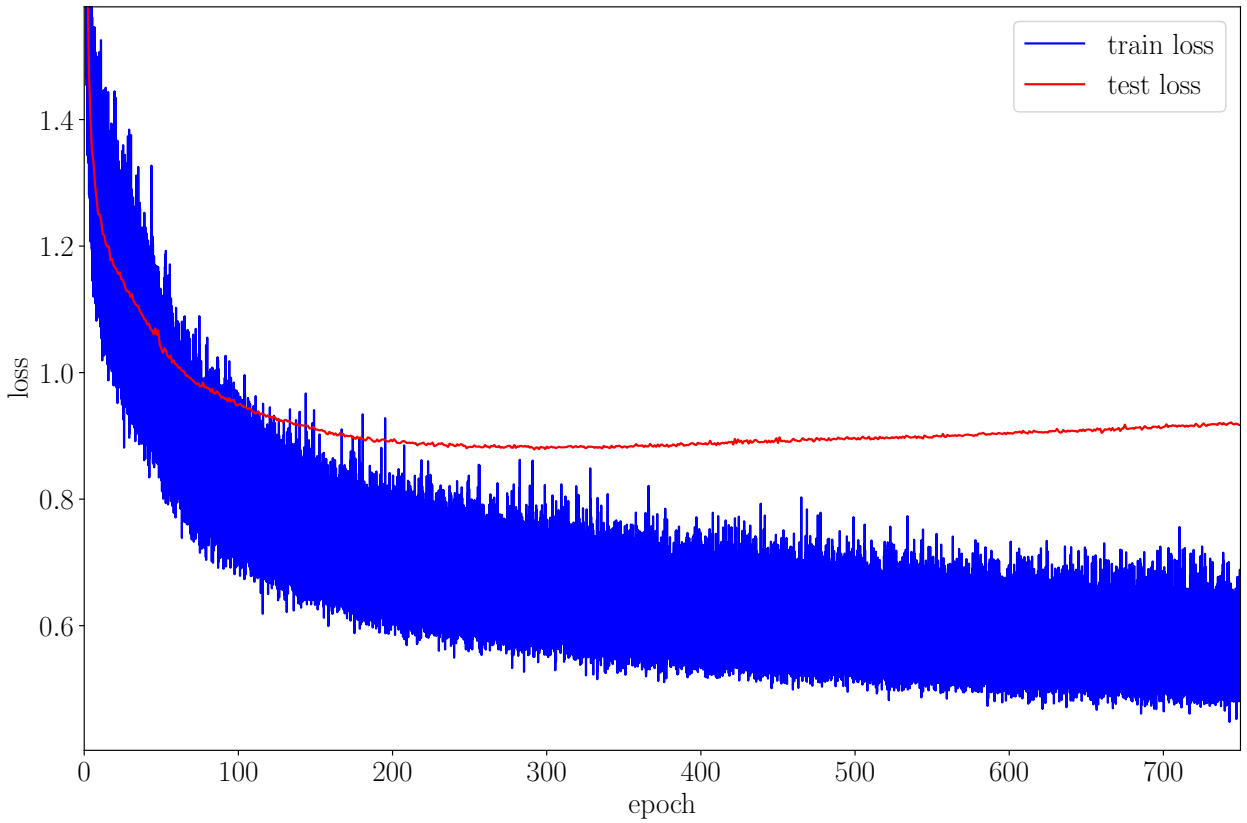

Figure 24: Visualization of our reconstruction MSE-loss that is overfitting. The optimum of the training loss is at the last epoch, but the validation loss rises after epoch 300. Nevertheless, we found the validation results of the last epoch to be better according to the histograms and the classifier test.

The first result from this study was that the number of neurons had rather to be increased than decreased. Even though the overfitting in the loss became worse, the reconstructions of the showers became significantly better for a setup of [10 000, 3 000, 500] neurons. The classifier AUC went down from 0.99 to 0.98 and the high level observables got closer to the GEANT equivalents (cf Figure 26, red vs green line).
Our ideas to prevent this overfitting did not work. We tried to use dropout and early stopping, but found that dropout was not helpful and that the reconstructions of the validation set were still improving during the apparent overfitting.
Our conclusion was that this overfitting was an artifact of a non-physical MSE loss. If our model is trained on showers with very many sparse voxels and if the model is expressive

enough to actually model the activity, we expect the VAE to exactly predict the correct position of every energy deposition of the training set. However, we cannot expect the same for the validation set because of the following two reasons:

1. We expect our model to learn trajectories of objects (showers, particles, etc.) that propagate through the layers. If this trajectory is slightly wrong all voxels that were originally hit by this objects are missed. But additionally, some other voxels receive this energy. Since they were likely not hit at all due to the generally low activity, we have two almost equal sized contributions to our loss function. This means that a trained model that predicts wrong trajectories can have a larger validation loss than an untrained model that always predicts the correct mean of each layer but only small voxel to voxel differences. As our layer energies are enforced to be correct, the second scenario is realistic for an untrained model. An example for this problem is visualized in Figure 25

2. There are symmetries in our data that are not captured by the MSE. While the network might be able to break this symmetry for the training set by memorizing something, it has to purely rely on the physics for the validation showers. This means that every symmetry will result in possibly wrong trajectories, that increase the validation loss.

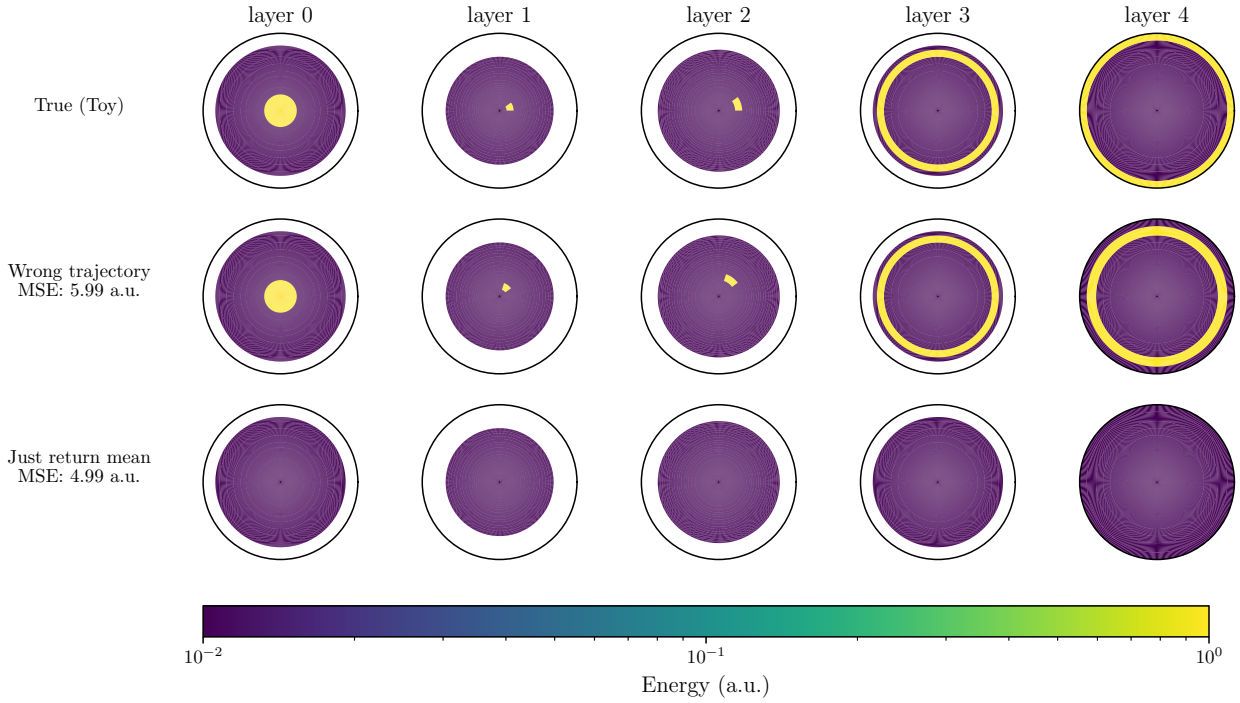

Figure 25: Visualization of the fundamental problem with a voxel-by-voxel comparison as it is done by the MSE. The "quality" estimate of random noise can be "better" than the corresponding estimate with a wrong trajectory.

As a result we tried to make the loss more physics compatible to reduce the punishment for slightly wrong energy depositions. So far we did not find a truly physical reconstruction loss

that incorporates all symmetries. If one would find such a loss the resulting reconstructions would probably be a lot better. However, we found some non perfect approaches that were helping as well.

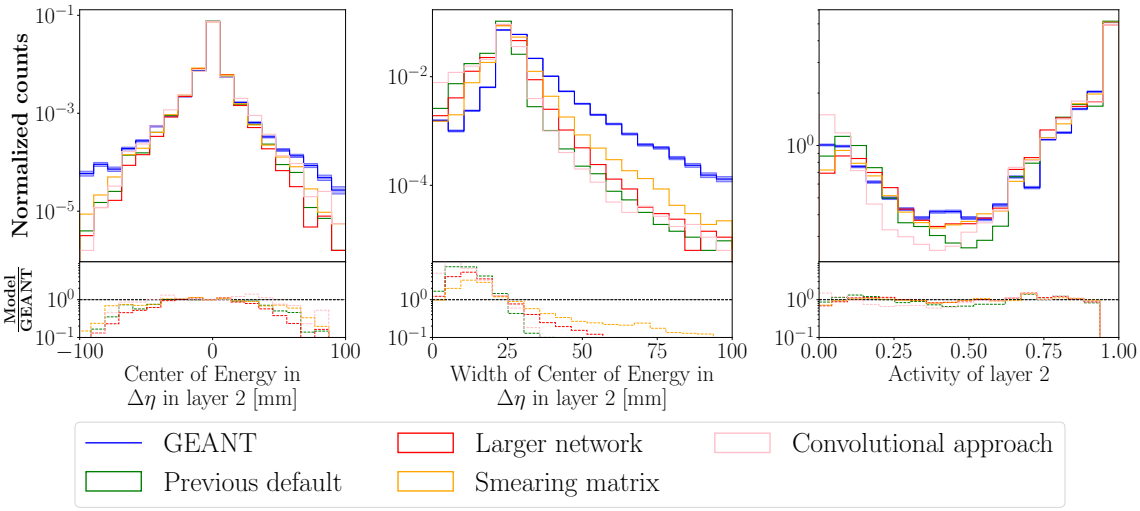

Figure 26: Visualization of the initial approaches to train the Gaussian VAE on the first CaloChallenge dataset.

Smearing matrix and convolutional approach To highlight the local properties of our showers we first tried to apply a slight blurring. After this blurring every energy deposition affects also the neighboring voxels. This way we are emphasizing the effect of clustered energy depositions, which should be physically more relevant. Furthermore, an energy deposition that is just one voxel off, is still overlapping after the blurring with the "correct" voxel. We found the best results when applying the blurring during training to the reconstruction and to the original, directly before the MSE loss.
However, as the binning in the first CaloChallenge dataset is not uniform, it was not possible to define a reasonable neighborhood property across layers. Because of this we applied the smearing as a 2D 3×3 convolution in each layer:

$$x_i \leftarrow s \cdot x_i + n \cdot \sum_{j \in \text{Neighbors}(i)} x_j .$$

Here we denote x_i as the energy deposition x in voxel i. For the neighborhood we chose periodic boundaries for the angles. Furthermore, we considered the opposite points in the innermost layer to be direct neighbors as well (cf. Figure 27). The voxels in the outermost layer had just 5 neighbors in this scheme. We found the best results for $s = 0.6$ and $n = 0.05$. The improvement, resulting from the smearing matrix is visible in Figure 26. The yellow line is closer to the ground truth for all used high level observables. Furthermore, the classifier AUC went down from 0.98 to 0.92 due to the addition of the smearing matrix.

After the success of this "smearing matrix" approach, we tried a convolutional approach to learn a more flexible smearing. We used small convolutional networks that operated on the detector layers with non-degenerate spatial resolution (i.e. layer 1 and layer 2) using the same neighborhood as before. The other layers and the conditions were processed individually using small fully connected networks. Afterwards these small networks were concatenated. Then, the concatenation was processed using the previous setup. The decoding step was designed analogous. We hoped that this model learns a smarter smearing matrix, however the results were not satisfying. We found it not only inferior to the simple smearing matrix, but also to our previous default setup (Figure 26). Therefore, we did not pursue the convolutional approach much further.

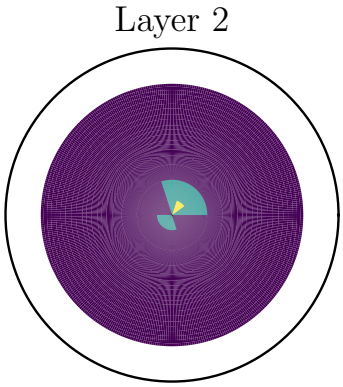

Figure 27: Visualization of the chosen neighborhood in layer 2. The bright voxel is the center, the dim voxels are its neighbors.

Learn decoder width Our last take on the overfitting was the try to learn a width for the decoder network. This way the network is forced to output an error estimate for every voxel of the reconstruction. We were hoping to learn which voxels the network was able to learn and which not. Checking if some region of our detector is responsible for our overfitting. During the training we were using only the data MSE-reconstruction loss since the widths had to belong to a single space. It is not reasonable to assume the same σ in the data and in the logit space. Furthermore, we had to disable the smearing matrix, as it would harm our uncertainty estimates by blurring the training space.

In the beginning we found a very unstable training for this setup. During the training the network was sending the widths to 0 and to ∞, crashing the training around epoch 8, before learning reasonable shower reconstructions. Therefore, the error estimates appeared to be not trustworthy. Because of this we continued with the classifier analysis, described in section 4.3.2, "Classifier weights".

After some time we revisited this approach and tried to clip the predictions of $\log(\sigma^2)$. For a cutoff of ± 10, the training was reasonably stable and the widths were not moving to ∞ or zero anymore. We tried to run the training with no loss weighting and with our old β

and γ values. We found that the reconstruction quality suffered a lot when not reducing the weight of the KL-loss. Reasoning, that we need the error estimates for results similar to what we received before, we decided to keep our β and γ parameters. Since we were not using a logit loss they represent only one degree of freedom, resulting in an effective suppression of the KL-term of 10^{-9}. The resulting shower shape reconstructions were actually better than before, but the activity reconstructions were worse. Both an effect of the absence of the logit loss term.
The $\log(\sigma^2)$ estimates are shown in Figure 28. One can see that the network learns to assign smaller mean uncertainties to the outer rings in general. This makes sense as the average energy deposition in these rings is much lower. An interesting outlier in this context is the layer 2 (the third layer as we use zero-based indexing for the layers). There the mean uncertainties are in general higher and the outer layer corresponds not to the minimum in the means.
By looking at the maximum σs we see that the network tends to predict a high possible uncertainty for the outer layer, that is apparently unusual (as the mean is significant smaller than that). Our explanation is that the network is sure if it deposits no energy in the corresponding outer voxel, but it expects a large mistake if it does. Here layer 2 is again different, the network is apparently always certain about the outer ring. We suspect that this is a sign of mode collapse. The outer ring in the second layer is that sparse that the network does not populate it at all, therefore it is never very unsure about it. Here, the most uncertain points correspond to the second-to-last ring. This also explains the relatively large mean: It can never be *really* certain, since the network knows, that it is missing something.
This problem also explains the problem with the shower-width observable. If the outer rings are underpopulated, the width distribution must be shifted towards smaller widths.
So, in the end we did not observe a solution to the overfitting, but a failure mode of our VAE. However as we were not able to run the setup without crashes at this point, we were not using this knowledge immediately. Instead, we implemented it in section 4.3.2, "Restructured noise layer" by using a scaling of the outer rings to make them more relevant in the loss.

As a final comment: Even though we were able to fix the instabilities during the training by the clipping, they were another hint that a Gaussian probability model is probably not the best choice for this task. Though, when encountering them we considered them to be "normal" as we heard that learning the decoder width is unstable in general (cf. section 5.3 in [87]).

Classifier weights Once we investigated the overfitting behavior of the VAE, we started to analyze its quality in more detail. Therefore, we not only trained a classifier to distinguish the reconstructions from the original showers but also looked at it weight distributions. Here, we focus on the smearing matrix approach as it was producing the first reconstructions for photons that resulted in AUCs significantly different from one. However, as it can be seen in the upper part of Figure 29, the weight distributions are still clearly separated. Furthermore, there is a significant amount of showers with weight 0 or ∞, corresponding to an absolutely certain classifier, that is invisible in our figure.
The bottom part of Figure 29 shows that the classifier is sensitive to the wrong bulk region

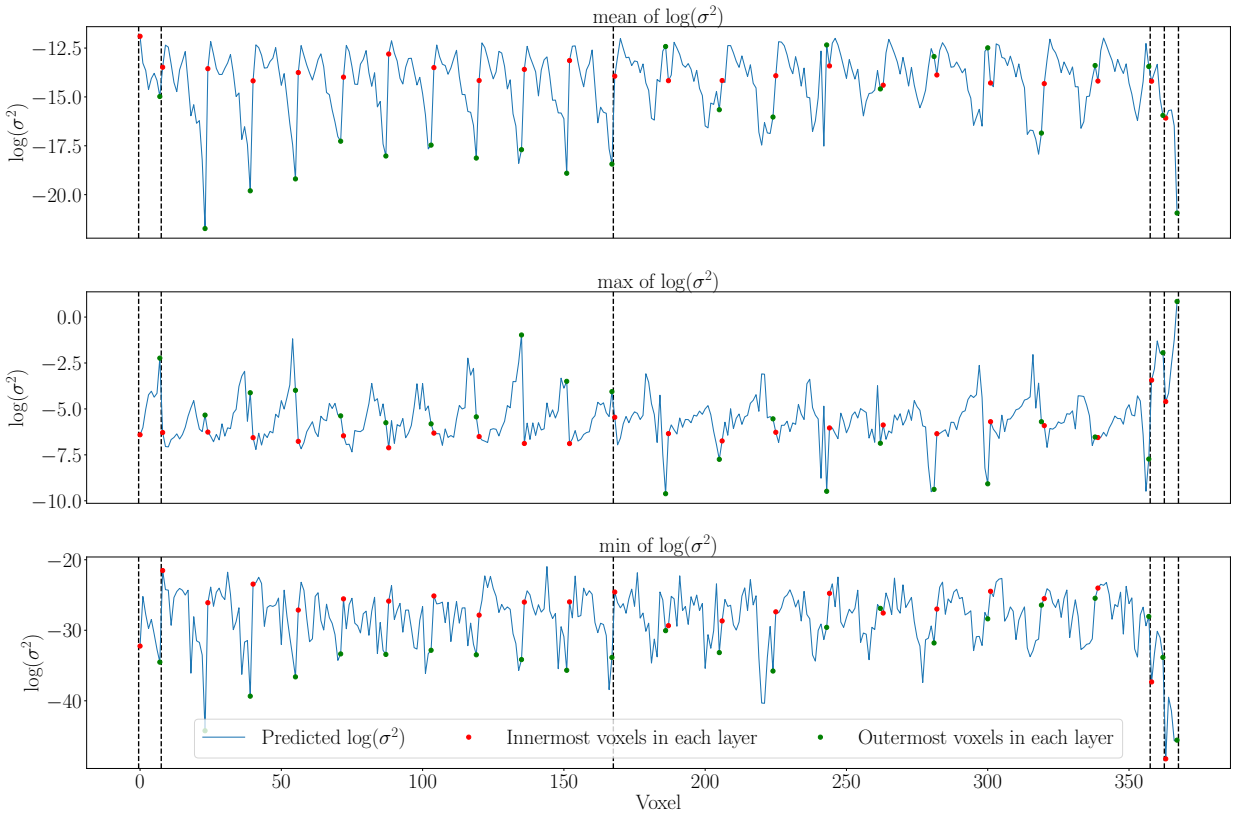

Figure 28: The log(σ^2) estimates of the VAE. The black dashed lines indicate the layer boundaries.

in the widths. Close to 0 mm, the small weight VAE density is significantly larger than the full VAE density. Implying that the classifier "says" that our network is oversampling the small width region. Before the classifier weight analysis, we considered this bulk the major issue as well, so this seems to be a reasonable choice made by the classifier. This also is in agreement with our (future) observation described in section 4.3.2, "Learn decoder width". Since we are under populating the outer rings, the showers are more compact, closer to the center resulting in a shift of the shower width plots.

Our second most relevant observable was the activity. However, the classifier is not sensitive towards this feature. The VAE was overpopulating the phase-space region, where all voxels are active. However, the low weight distribution is very low in this region, indicating that the classifier considers these showers to be "realistic".

This seems to be a fundamental problem with our classifier setup, as we made this observation frequently. The classifier seems to have trouble "seeing" differences in the high-activity regions. Our only solution so far is to look at a normalized classifier as well. The term normalized classifier refers here to the preprocessing of our data, where each voxel is normalized by the corresponding incident energy. The normalization seems to emphasize the effect of the activity, making it more obvious to the classifier. This matches with our observation that the normalized classifiers usually favor the logit loss term more than the unnormalized

classifiers: The are more sensitive to the activity and less sensitive to the logit loss induced small peak shifts.

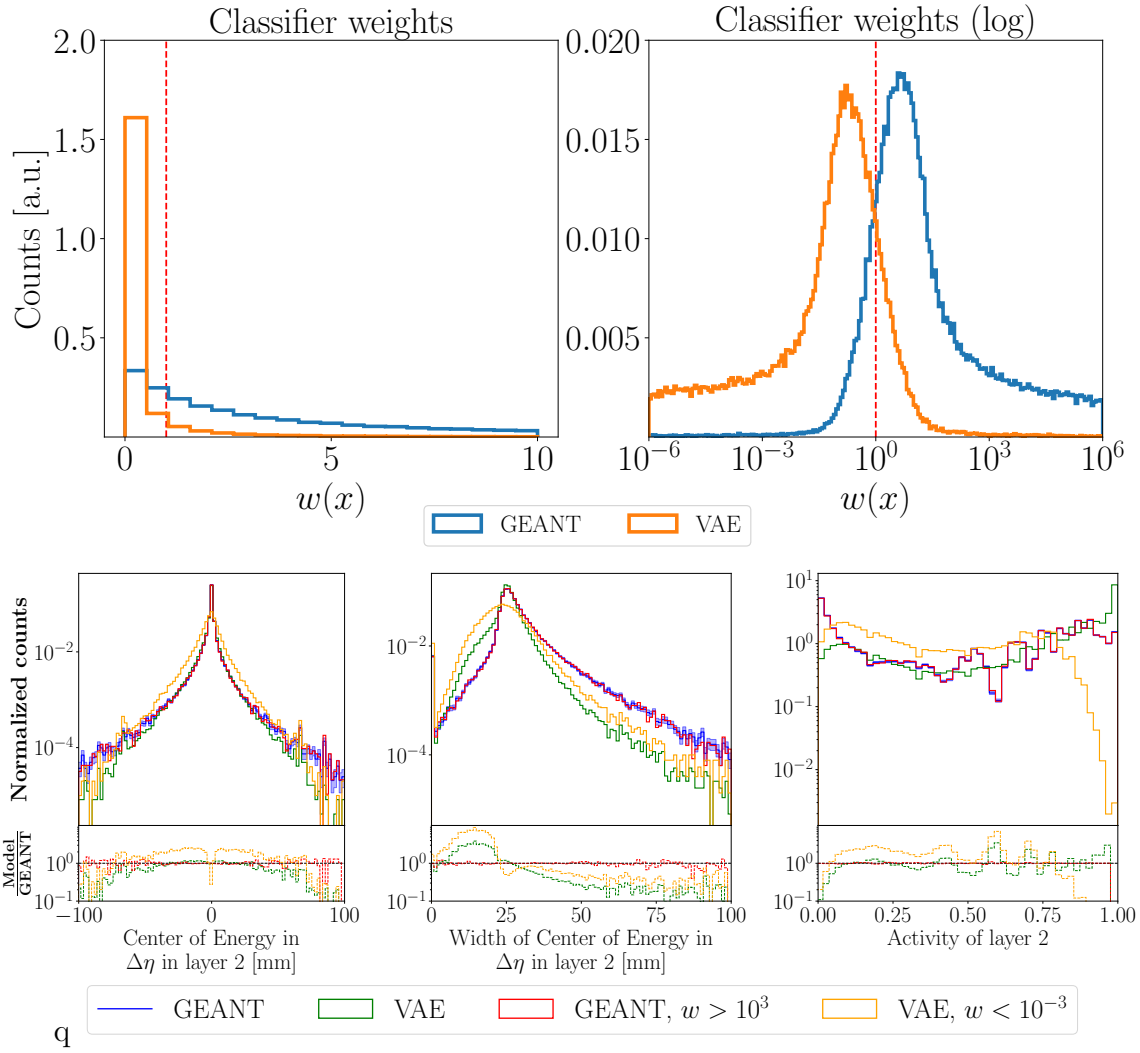

Figure 29: Results of the classifier weight analysis.
(Top) Weight distribution of the classifier.
(Bottom) High level observables for large and small weights. Layer 1 and 2 are very similar.

During the weight analysis we also tried to filter the reconstructions by their incident energy. This revealed an interesting problem with our model. As it can be seen in Figure 30 our network performs much better on higher incident energies. On lower energies, we see shifted peaks and an increasing bulk in the width plots. This means, that our problem with the wrong width plots is likely caused by the low energy particles. This makes sense from a physical perspective as well: The low energy showers have a larger fraction of "noise voxels". As our VAE is trying to reduce the dimensionality in a deterministic manner, it cannot simulate true noise. However, optimistically, we expected to not have reached this

hard boundary, yet. Therefore we started to analyze the low energy incident energies more carefully. To achieve higher computational efficiency we decided to reduce our training set to the relevant incident energies. We realized that the reconstructions got worse this way, but that the failure modes did not change. Therefore it was not only faster to train this setup, it was also emphasizing our problem. Furthermore, the training on individual energies prevented the artifacts from the other energies, enabling us of a more systematic analysis.
Of course one should use the full VAE for final deployment as the increase in training data will improve the overall results.

Noise and α investigation Since we were looking for a problem related to small energies, we decided to systematically investigate the behavior of the VAE under modifications of the noise and the logit regularization parameter α. After all, both correspond to numerically small values added to our data, that could overshadow small voxel entries. This effect would be stronger for small incident energies as the average energy depositions would be lower.

We started investigating the effect of the noise on the logit space. We looked at the energy distribution in one voxel. More precisely, we investigated how the distribution of x_{logit} for ($x <$ noise width) and ($x >$ noise width) behaved. These two distributions do not have to be separated in the logit space because of the event dependent normalization step. In fact we found the ($x >$ noise width) peak to be completely overshadowed by the ($x <$ noise width) peak for outer voxels at small incident energies. For larger incident energies or inner voxels this did not happen.
We tried to reproduce this overshadowing for higher incident energies by masking, discarding showers in the ($x >$ noise width) peak. However, the effect was only visible for the voxel that was used to compute the masking, it did not generalize over the full outer ring. However, we found that the masked high incident energy reconstructions did not became significantly worse. Even though this approach was not very clean, we were able to conclude that the voxel-to-voxel correlations were enough to separate the "noise" and the "no-noise" peaks in the full space. Otherwise at least the reconstruction of the masked basis voxel would have been worse.

After this we analyzed the effects of our α-regularized logit transformation on the low energy voxels. In some way we can see α as the scale in the normalized space to which the logit is sensitive. If an energy deposition is significantly below α, the logit will not return the order of magnitude of the deposition but $\log(\alpha + x_{0-1}) \approx log(\alpha) + \frac{x_{0-1}}{\alpha}$. Implying that the logit transformation looses its sensitivity to values (significantly) smaller than α. Since we are adding the α-value in our normalized space its effect is stronger for very active layers than for very sparse layers.
If only two voxels are active our "accuracy", determined by α, is not a problem as long as $\frac{\text{dim voxel}}{\text{bright voxel}} > \alpha$. Meaning that α describes how small the energy deposition of the dim voxels in relation to the leading voxel is allowed to be in order to be resolved by our setup.
However, if many voxels are active, the norm is affected by many voxels. So, if all voxels are active, we receive the following inequality constraint for our sensitivity:

$$\frac{\text{dim voxel}}{\sum \text{bright voxels}} > \alpha.$$

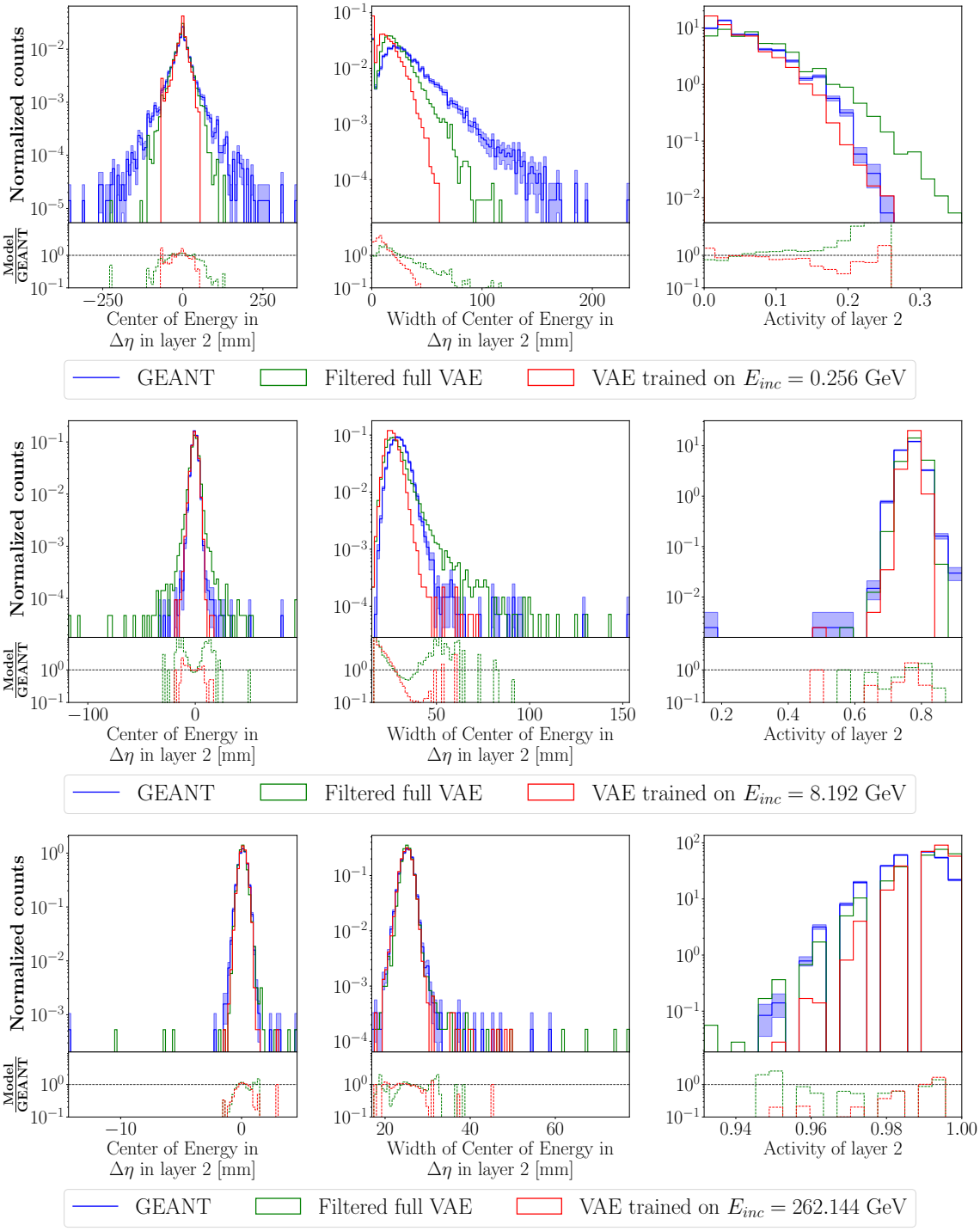

Figure 30: High level features, when looking only at some fixed incidence energy. Here we show the lowest, one intermediate and the largest one that does not suffer from less statistics. One sees clearly that the VAE performs better on high incident energies. Additionally we show the results of a VAE that was only trained on the corresponding incident energy. The reconstructions are worse, but the failure modes are the same.

In the worst case scenario of one dim and $N-1$ equally bright bright voxels this would imply

$$\frac{\text{dim voxel}}{\text{bright voxel}} > (N-1) \cdot \alpha,$$

where N is the number of (active, bright) voxels in the layer. This also means that for very large layers, the possible range between small and large voxel, that can be seen by the network, shrinks.

Because of this observation we realized that it will be necessary to reduce α for the larger datasets 2 and 3. Therefore, we tried to reduce α as much as possible to learn out about the low-α effects. We found numerical problems for $\alpha < 2.9803 \cdot 10^{-8}$. The reason is that for python numbers (and for float32 PyTorch tensors) the following equation holds due to numerical reasons for $\alpha < 2.9803 \cdot 10^{-8}$:

$$1 - \alpha \equiv 1.$$

This means that the logit is not a true "two-sided logarithm" since it is unable to resolve values close to one the same way it is resolving them close to zero. It is predicting infinities to early.
One should note here that this is not an inherent problem of the logit but of the needed accuracy for the values close to one and thus of the float format itself. It might be possible to solve the problem by storing $1 - x$ instead of x as a float, but we found that the values close to one were not needed to be resolved that well at all.
We tried to split the α parameter in two contributions, a shift α_0 and a scaling α_1, resulting in the following regularized logit transformation:

$$\text{logit}(y) = \log\left(\frac{y}{1-y}\right), \qquad y = \alpha_0 + x \cdot (1 - \alpha_1).$$

Empirically, we found that the parameter α_0 was very important for our reconstructions. Using a too large value here was significantly worsening our results. However, the α_1 parameter was not very relevant. We were able to increase it to 0.1 without observing worse reconstructions. This observation enabled us to reduce the important part of α, the shift, at will, without creating instabilities because of the scaling.
One immediate success from this investigation was that we were able to understand and solve the numerical problems that were mentioned in section 4.3.1, "Try noise". For evaluations of the type of logit (unnormalize (sigmoid (logit (x_{0-1})))) we can reach values very close to 1 in the outermost logit call due to our normalization layer. So, it could happen that this concatenation creates infinities during the gradient computation, even though we chose $\alpha > 2.9803 \cdot 10^{-8}$.
To prevent this problem we chose a large value of 0.1 for α_1 from now on. Since this change we did not observe the unstable loss values for the rest of our experiments.

Restructured noise layer Once we understood the reasons for the instabilities described in section 4.3.1, "Try noise" we tried to use the more straightforward noise implementation

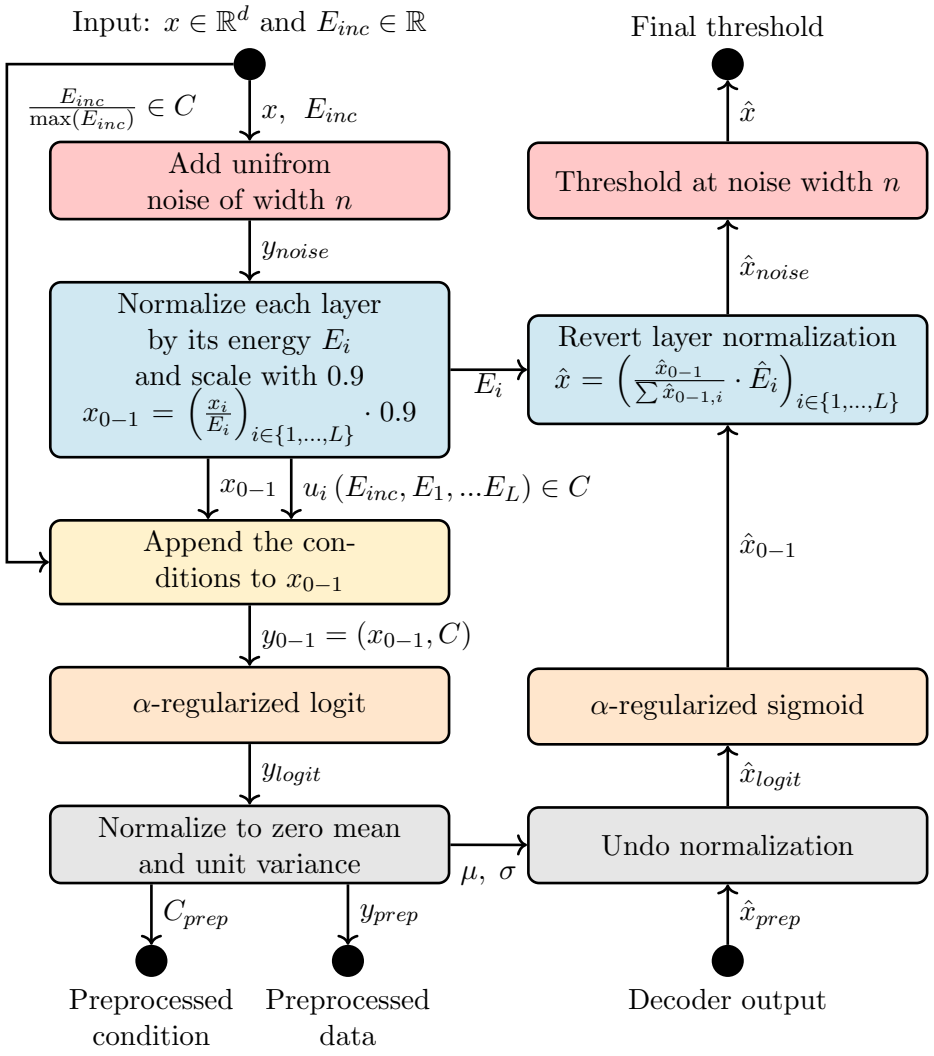

Figure 31: Summary of the second version of the VAE-preprocessing with the new noise layer structure.

that was described there. The new preprocessing is depicted in Figure 31.

For stability, we used $\alpha_0 = 10^{-6}$ and $\alpha_1 = 0.1$. Our network is now always using the full forward pass, consisting of preprocessing, encoder, decoder and postprocessing, except for the final threshold. Furthermore, we used our knowledge from section 4.3.2, "Learn decoder width" at this point by using a different normalization for our data loss. Instead of the layer energy we started to use the mean energy of each voxel, over all showers. This enabled us to increase the importance of the outer rings, resulting in better overall results.
For the logit loss we applied the normalization and the regularized logit transformation to the reconstructions and the original showers, before the final thresholding, resulting in the following loss:

$$\mathcal{L}_{\text{ELBO}} = \gamma \cdot \text{MSE}\left(\frac{x}{< x >_{\text{showers}}}, \frac{\hat{x}}{< x >_{\text{showers}}}\right) + \delta \cdot \text{MAE}\left(x_{logit}, \hat{x}_{logit}\right) - \beta \cdot \sum_i \left(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2\right).$$

Additionally, we were using a very small weight decay of 10^{-7}. We did not observe any different results because of it, but we wanted to prevent a possible degeneracy. Since the network does not see any effects of the scale of its output - we renormalize it anyway - it might end up producing very large outputs, which could produce instabilities. A small weight decay should prevent this degeneracy.
These changes had a significant impact of our reconstruction quality. The most important effect is the improved energy deposition per layer for low energies. Since we were are not neglecting any small values in our norm, we were receiving a "clean" cutoff in the voxel energy distribution by default. So, there was no reason to normalize to the distorted layer energies anymore. Because of this we were able to reconstruct lower layer energies. Nevertheless, one should note that the noise thresholding after the normalization results in a global shift of the layer energy distribution. This could, again, be fixed by normalizing to the distorted layer energies. However, for this setup it seemed to have a negative overall influence.
Furthermore, we found all plots to benefit from the new structure, at least a bit. Also the classifier AUC went down from 0.92 to 0.88. Some high level observables, visualizing the described effects can be seen in Figure 32.

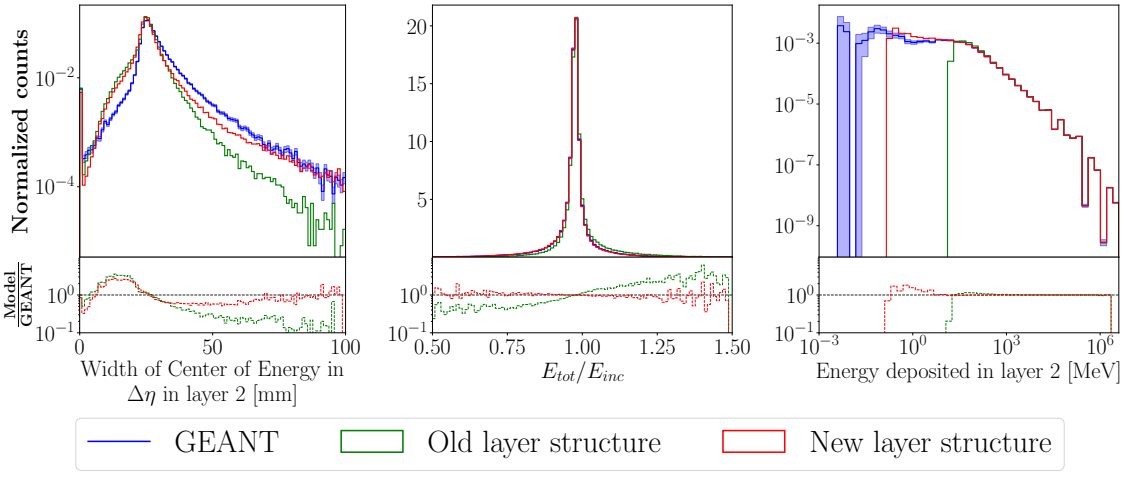

Figure 32: The effect of the new noise structure. The energy related histograms profit the most, but all other observables seem to benefit from the new setup. Especially the tails are reconstructed better.

A positive side effect of the new setup is that we found a possibility to estimate γ during the training. With the new setup we found that the logit loss and the data loss were numerically of the same order of magnitude for a good choice of γ. Driven by this observation we tried to choose gamma after 20 epochs such that this is the case. This approach was working,

potentially reducing the γ-dependence a lot.

As we were mainly looking at the individual incident energies at this point, they should be discussed as well. To reduce the number of plots we will show only the histograms for the lowest incident energy of 256 MeV (Figure 33), as it was the most problematic one. However, the experiments were done also for the incident energies of 8 GeV and 262 GeV.
Since we were not thresholding in the logit space anymore, it became possible to not only threshold, but also to subtract the noise. Doing this improved the shower widths and the layer activities. However, a similar gain was achieved by just subtracting the average noise. Since the subtraction was worsening the E_{tot}/E_{inc} histogram, we decided to add a final shift after our noise subtraction. In this last shift, we enforced this observable to be correct, by applying a global normalization to the full shower.
The last attempt to improve the results of the Gaussian VAE was to find a way to reduce the needed noise level. We tried to reduce it from $n = 10^{-6}$ to $n = 10^{-8}$. This turned out to be more difficult than expected. We were only able to receive a stable training, when using batch normalization. We found that the low-noise batch norm results improved the activity and the low energy regimes of the individual layers. However, the shower shapes, especially their tails, were consistently worse. Also the classifier had an easier time with the batch norm variant. Some high level features for the batch norm approach can be seen in Figure 34
At this point, we thought we were hitting a wall and started looking for alternatives to a Gaussian VAE. The main reason was that most of our approaches were improving our results a bit, while none was *solving* our problems. It felt like we were only fighting symptoms. Motivated by the results from Dalila Salamani's thesis [88], we decided to try a Bernoulli VAE, replacing the MSE loss with a BCE loss (cf section 3.3.7, "Bernoulli decoder").

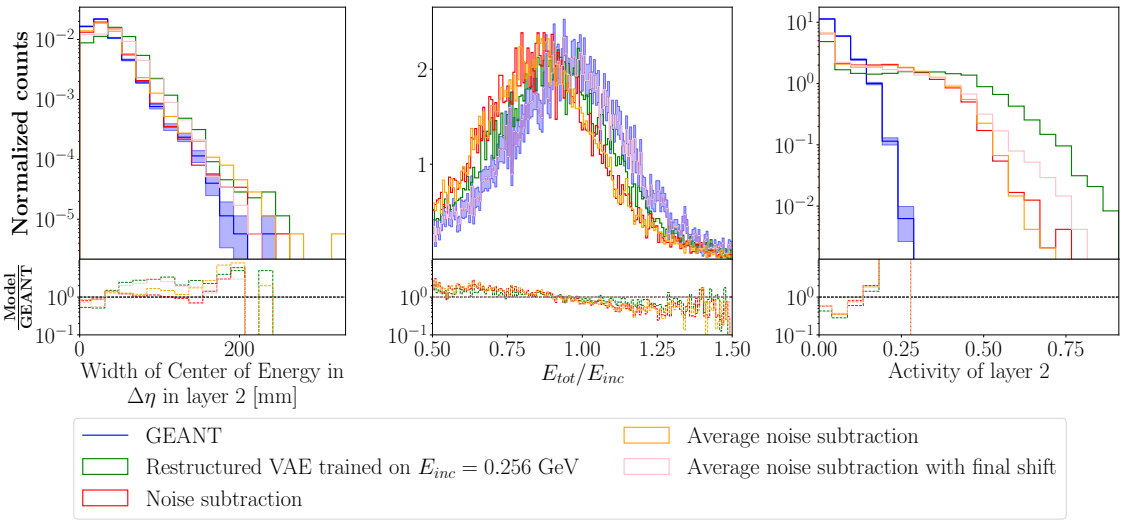

Figure 33: Results of VAEs that were only trained on the lowest incident energy of 256 MeV

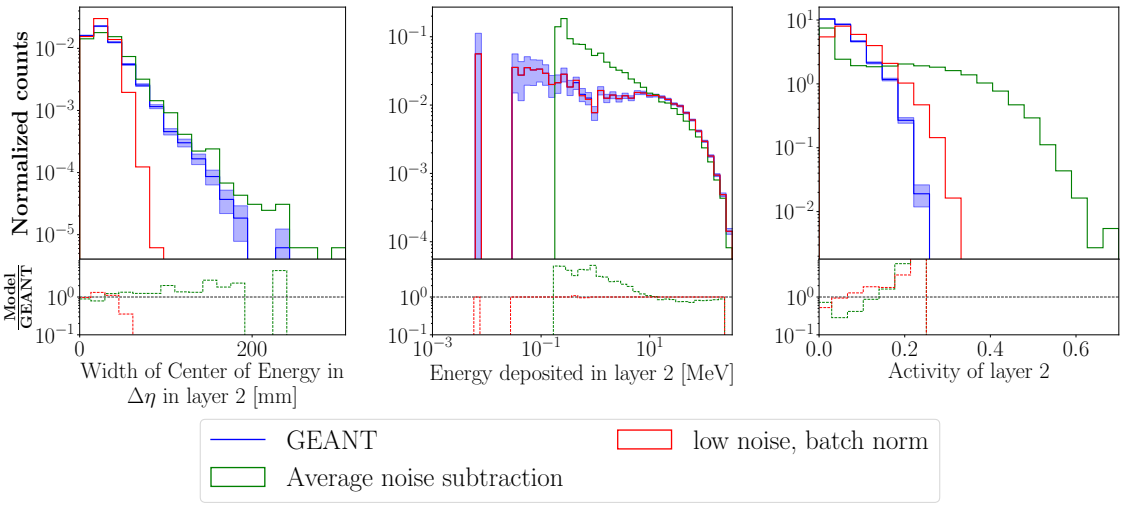

Figure 34: Results of VAEs that were only trained on the lowest incident energy of 256 MeV. Here, we are comparing a low-noise batch-norm variant to the version with larger noise and noise subtraction (without final shift).

4.4 Bernoulli VAE

4.4.1 CaloChallenge dataset 1

Activity loss The reconstructions of the Bernoulli VAE without logit loss and without noise were already better than any reconstruction achieved with a Gaussian model assumption (Figure 36, green line). In fact, we found that adding noise was not helpful at all. However, we realized that the thresholding was helpful. Therefore, we decided to use the thresholded reconstructions for the logit and the BCE loss. Furthermore, it turned out that the underlying NN can be smaller, when using a Bernoulli approach. Here, we ended up with 5000, 1000, 500 hidden neurons. Resulting in a six times smaller network size. Last, but not least, we found the smearing matrix to have a much smaller effect on the Bernoulli VAE. Using it was slightly improving the shower shape observables but worsening the activity observables. This is reasonable, since the BCE is using the logarithm, internally. This means that it is more sensitive towards small energy depositions. Adding bin migration from neighboring voxels weakens this sensitivity towards the low energy regime of our data. Therefore, we decided to not use the smearing matrix for the Bernoulli VAE.
The only downside was the fact that the activity was not well reconstructed, even without the smearing matrix. So, we decided to continue with the BCE loss instead of the MSE loss and tried to enhance the bad activity of our reconstructions. First, we tried to add a logit loss, computed the same way as before. Using it, we found the best results when the logit loss was about two orders of magnitude smaller than the BCE loss. Interestingly, this point corresponds again to $\gamma = 10^4$, the same value as before. The attempt to increase γ beyond this value was worsening the reconstructions.
Since the activity was still not perfect, we tried to add a sparsity inducing loss as well. Since we knew already that a naive activity loss would not work (cf section 4.3.1, "High level loss"), we tried to find a differentiable activity measure: Instead of adding step functions $\theta(x)$ we

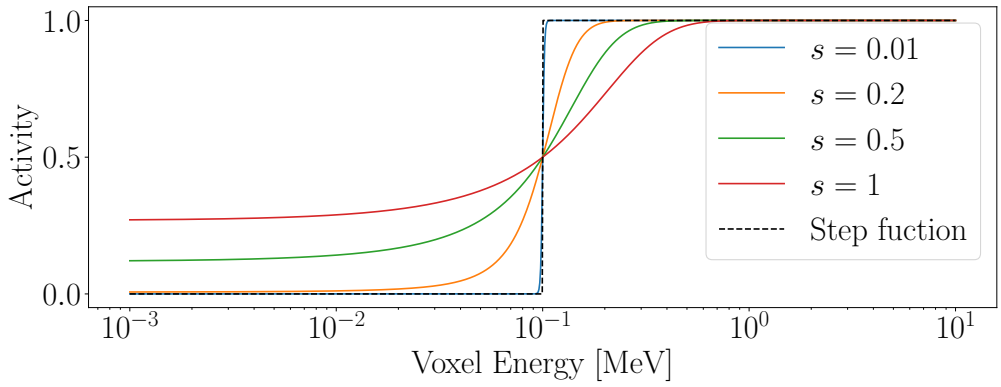

Figure 35: Visualization of our activity approximation for one single voxel. For the total activity we simply take the mean over all activities. For larger s values the activity estimate is not approaching 0 since our x-axis is log-scaled and thus positive.

decided to add up sigmoid functions:

$$\text{activity} = \sum_{v \in \text{Voxels } V} \frac{\theta(v > t)}{|V|} \approx \left\langle \frac{1}{1 + \exp\left(-\frac{v-t}{s \cdot t}\right)} \right\rangle_{v \in V}$$

Here, t is the corresponding activity threshold and s the strength of our sigmoid. For our experiments we used $t = 10^{-1}$ MeV and $s = 0.2$. We chose t to be different to the "proper threshold" as defined in section 3.4.3, "CaloChallenge dataset 1", as this allows us the verify the generalization power of the activity loss, beyond the actual point of the threshold. s was chosen such that our sigmoid was increasing from 0.01 to 0.99 within two orders of magnitude. To use the activity loss, we added a MSE comparison of the activity estimate of reconstruction and ground truth. We scaled this activity-loss term by a factor of 1000. We tried different values for this scaling between 0.1 and 10000 and found the best high level observables with it.

Even though we found better high level observables for our reconstructions (Figure 36, yellow line), when learning with activity and logit loss, it seems that these additional loss terms where also introducing some unseen artifacts. In fact, it was much easier for a classifier to pick the reconstructions of a VAE that was trained with the additional loss terms, when using all incident energies. With the extra losses, we were receiving an AUC of 0.92, using only the BCE this value went down to 0.84.
Because the pure BCE loss is closer to the underlying theory and because of the previous observation, we decided to keep only the BCE loss and not the logit or activity loss.

Why use a Bernoulli model The next question that we were analyzing, was why the Bernoulli model was more suited than the Gaussian model. After all, the Bernoulli distribution is discrete, while our data is continuous. We found two answers to this question.
First of all, one has to consider that also the Gaussian model needs a strong assumption. Namely, that our energy depositions are following a normal distribution. Apparently, this

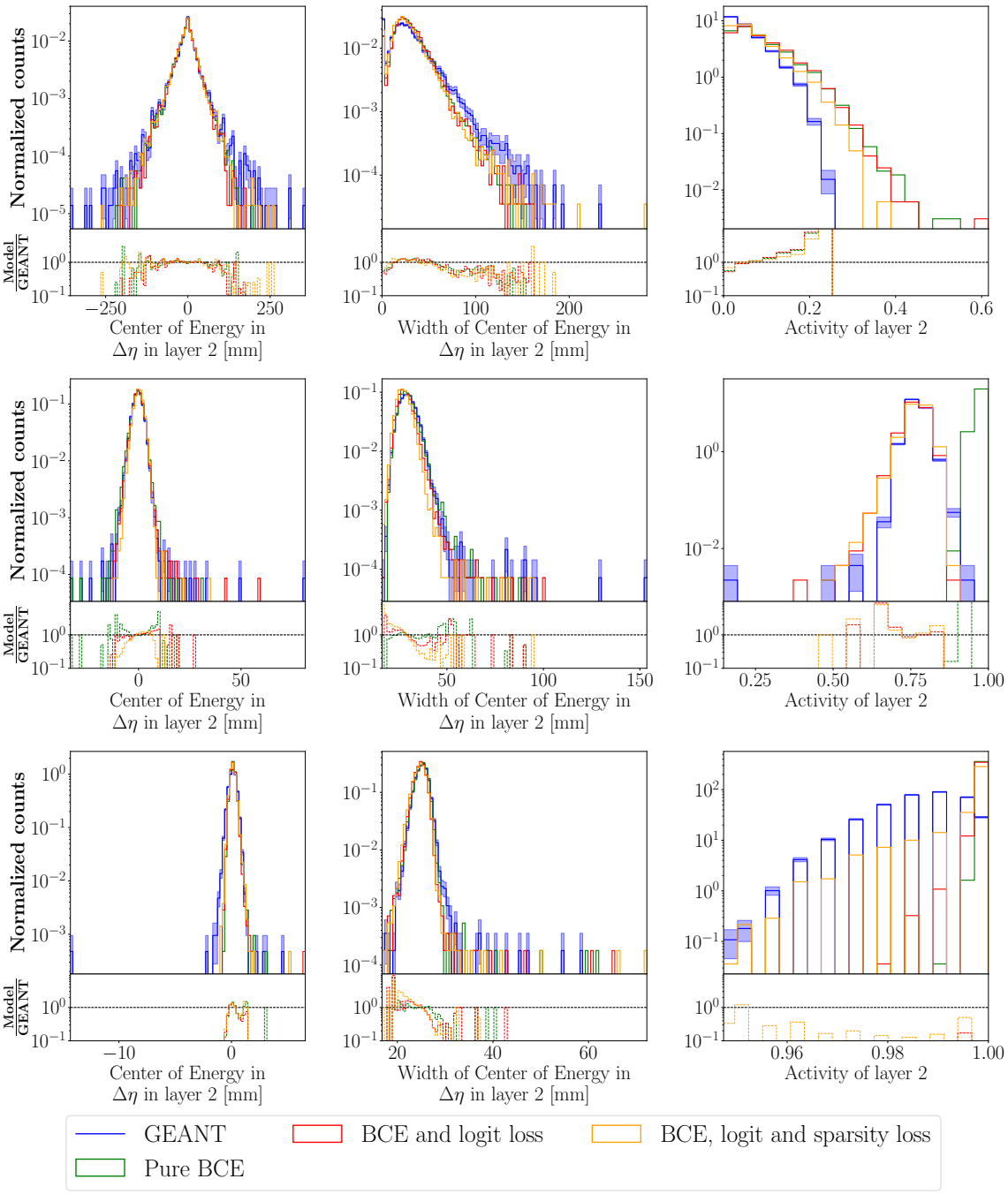

Figure 36: High level features, when looking only at some fixed incidence energy. The loss setup with activity and logit loss seems to produce better reconstructions for the activity-observable. However, the shower shapes seems to become slightly worse. The setup with only activity loss without logit loss is not working at all. It is worse than all the other approaches. It seems that the activity loss needs the logit loss to generalize beyond the actual point of its threshold.
Top: 256 MeV, Mid: 8 GeV, Bottom: 262 GeV.

assumption is "more" false than the assumption of a discrete Bernoulli distribution. In fact, our data is very similar to discrete values in the normalized space.
Our second explanation is based on the gradients of the BCE. As one can see in the right plot of Figure 37, the MSE suffers from stagnating gradients when the prediction is much closer to zero than to the target. On the other hand, the BCE gradient has a pole of the order x^{-1} around zero, leading to large gradients in this situation. This means that the BCE has a bias to produce too active layers, in line with what was previously seen. However, this has an advantage during training. If most voxels are active, the network receives more information since fewer voxels are muted by the ReLU. This enables faster and more stable training.
This hypothesis was tested by using a heavily finetuned preprocessing for the Gaussian VAE, where we were trying to assimilate this gradient behavior of the BCE for the MSE. More precisely, we used the preprocessing function

$$f(x) = x - \frac{1}{x} - 10 \cdot \log(1 - x)$$

and finetuned alpha and noise values for each separate incident energy. The effect on the MSE loss is visible as the yellow line in Figure 37. In the end we were receiving a similar reconstruction quality to the Bernoulli VAE. However, this approach is not practical as this amount of finetuning is unrealistic in any practical scenario and we did not pursue this finetuning possibility further.

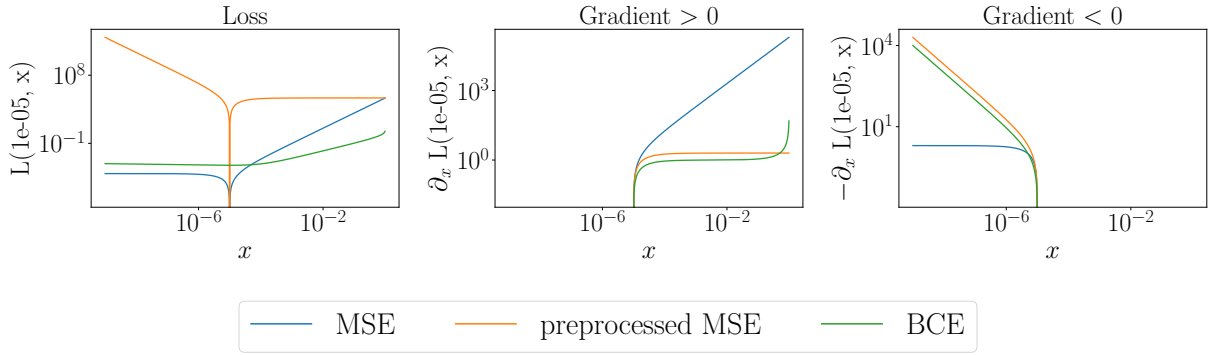

Figure 37: Comparison of the BCE and the MSE loss and of their gradients.

Continuous Bernoulli Even though we argued before, that the discrete Bernoulli distribution is a better assumption than the normal distribution, it must not be true that there is no better model one could assume. Proceeding with this thought we were expecting even better reconstructions using the continuous Bernoulli model (section 3.3.7, "Continuous Bernoulli decoder"). However, using the (pure) corresponding loss function, resulted in very unstable training. In fact, the network was not learning anything as long as we tried to learn the normalization constant $\log(C)$. The reason for this "muting" of our voxels is easy to explain by the fact, that we do not want to be discrete, even though we are close to it. The difference between a 10^{-1} MeV and a 10^{-2} MeV voxel is of huge importance for

us. By making our data discrete we are simply loosing this information. This problem is made even worse by considering that we are not applying the loss in the unnormalized space but in the normalized space. Thus, we have to reconstruct faithfully all orders of magnitude between $O(1)$ and $O(10^{-6})$, as we want to be at least sensitive down to our α regularization. Assuming discreteness here is simply not correct.
However, the normalization constant C of the continuous Bernoulli has exactly this effect. Its gradient has a very strong attractive divergence of strength $> \frac{1}{x}$ at zero and one (cf. Figure 38). Therefore, this the network is just learning to produce zeros as much as it can. Because of this downside, we are using the discrete Bernoulli model instead of the continuous Bernoulli model.
In Figure 39 we show the average shower as predicted by a VAE that we trained initially for 50 epochs without $\log(C)$. Afterwards, we increased the relative strength of the $\log(C)$ loss over the range of 50 additional epochs linearly to 1. As one can see, only layer 0 was not collapsing at this point as it has the largest average energy of all the layers, so the $\log(C)$-divergence has the smallest effect on it. Nevertheless, when training longer it will be "muted" as well.

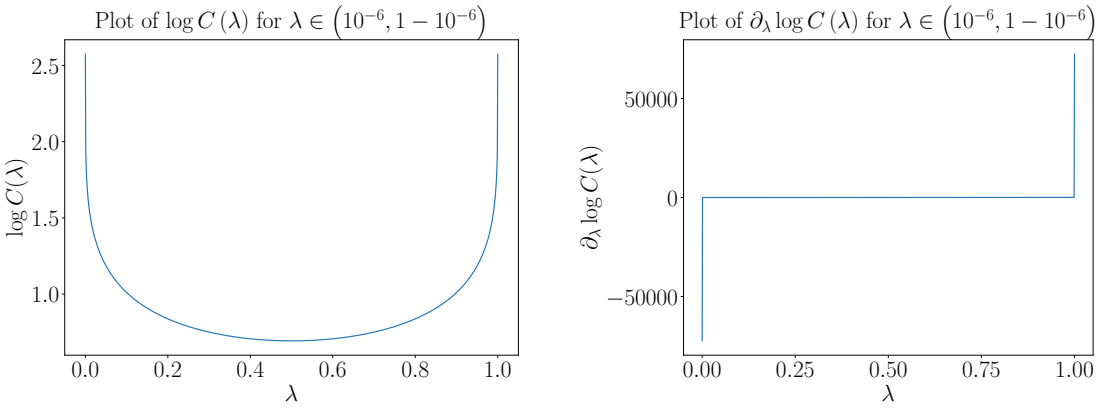

Figure 38: Visualization of the normalization constant of the continuous Bernoulli VAE and of its problematic gradient.

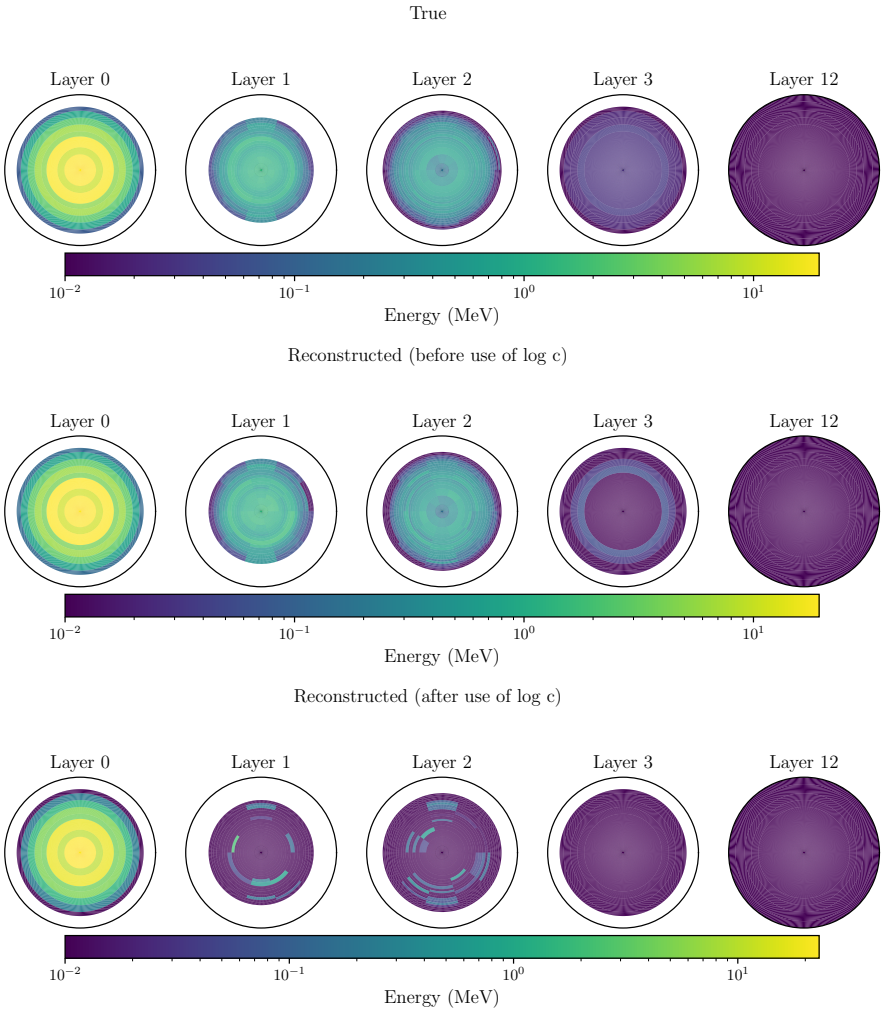

Figure 39: (top) True average shower for 0.256 GeV
(mid) Reconstructed average shower for 0.256 GeV (Without $\log C$)
(bottom) Reconstructed average shower for 0.256 GeV (With $\log C$) in increasing strength for additional 50 epochs

Final results dataset 1 At this point we concluded our optimization of dataset 1. We ended up without noise and with the discrete Bernoulli model. Furthermore, we have decided to not use any further losses besides the plain BCE. A detailed list of all hyperparameters for the CaloChallenge dataset 1 setup can be seen in Table 5. In Figure 42 and Figure 43 we show all interesting high level photon observables for the VAE reconstruction and the VAE sampling with an INN. Furthermore, we added a comparison to pure INN results, using the hyperparameters in Table 5, "pure INN (ds 1)". One can see that everything besides the layer activity seems comparable. Nevertheless the INN is still "better", as long as it can be applied.
The plots in Figure 44, Figure 45 and Figure 46 show the equivalent results for the pion dataset. Even though, we did not use it to tune the network and its hyperparameters, they seem to be of equal quality.
The weight distributions can be seen in Figure 40 and Figure 41. We show only the distributions for the generated samples and not for the reconstructions as we are more interested in the generation power at this final point. We see that the weight distributions are only "good" for the photon INN samples. All of the three other cases result in clearly separated weight distributions. Interestingly, we are almost reaching the generative power of the INN for the more complex pions samples. Implying, that the VAE approach works better for hadronic showers than for electromagnetic showers.

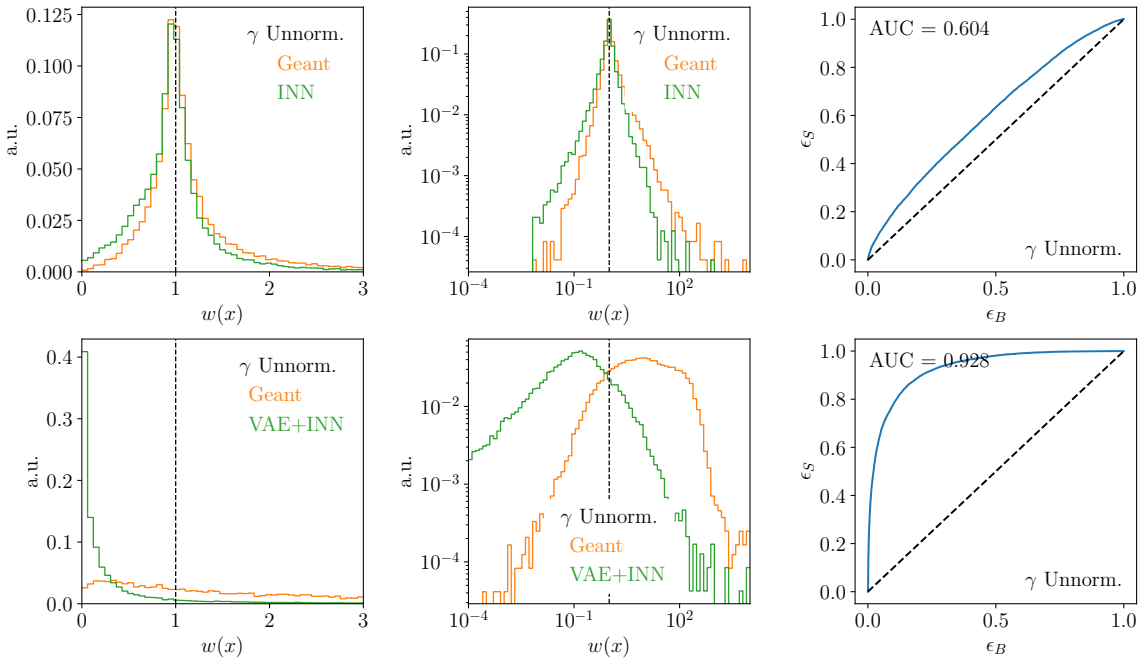

Figure 40: Classifier trained on photon showers.
(top) The results for the pure INN.
(bottom) The results for the combination of VAE and INN.
Weights distribution in linear space (left) and in log-space (center). (right) ROC curve and relative AUC score.

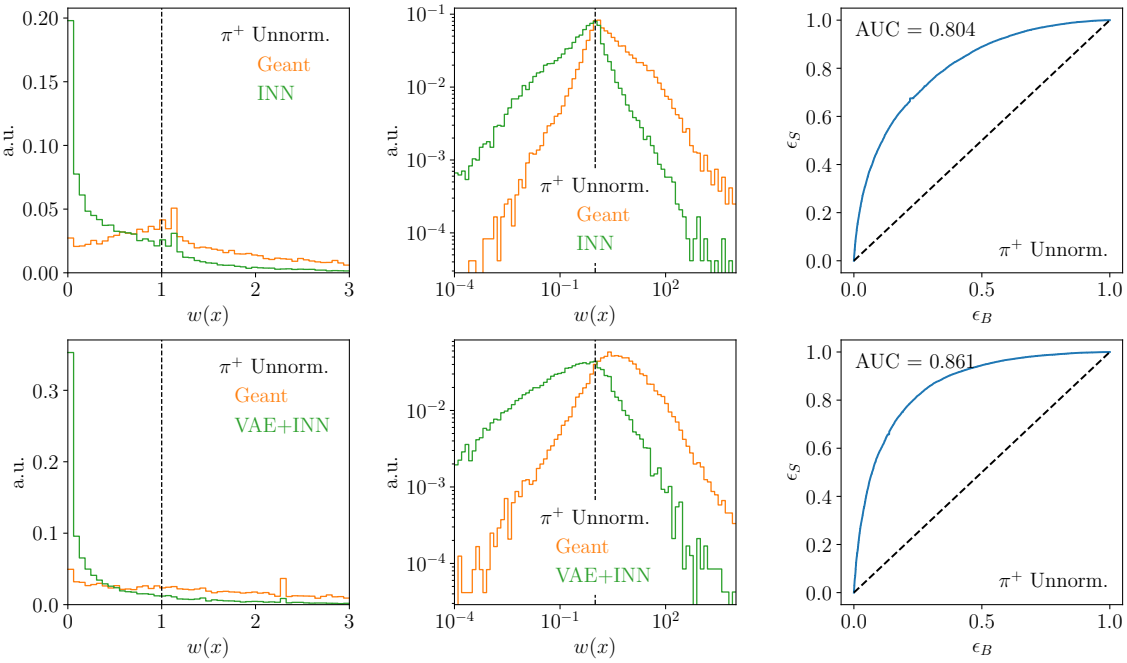

Figure 41: Classifier trained on pion showers.
(top) The results for the pure INN.
(bottom) The results for the combination of VAE and INN.
Weights distribution in linear space (left) and in log-space (center). (right) ROC curve and relative AUC score.

Parameter	Pure INN (ds 1)	Pure INN (ds 2)	INN (After VAE)
coupling blocks	RQS	Cubic	RQS
# layers	4	3	3
# of hidden neurons	256	256	32
# of bins	10	10	10
# of blocks	12	14	18
# of epochs	450	200	200
batch size	512	256	256
lr scheduler	"one cycle"	"one cycle"	"one cycle"
max. lr	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
n	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	0
α	$1 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$1 \cdot 10^{-6}$ (Set 1)
			$1 \cdot 10^{-8}$ (Set 2,3)

Parameter	VAE	
lr scheduler	Constant LR	Inner VAE
lr	$1 \cdot 10^{-4}$	
hidden dimension	5000, 1000, 500 (Set 1)	
	1500, 1000, 500 (Set 2)	
	2000, 1000, 500 (Set 3)	
latent dimension	50 (Set 1,2) / 300 (Set 3)	
# of epochs	1000	
batch size	256	
γ	$1 \cdot 10^{4}$	
β	$1 \cdot 10^{-5}$	
hidden dimension	1500, 800, 300	Kernel
kernel size	7	
kernel stride	3 (Set 2), 5 (Set 3)	

Table 5: Network and training parameters for the different models used for the CaloChallenge.

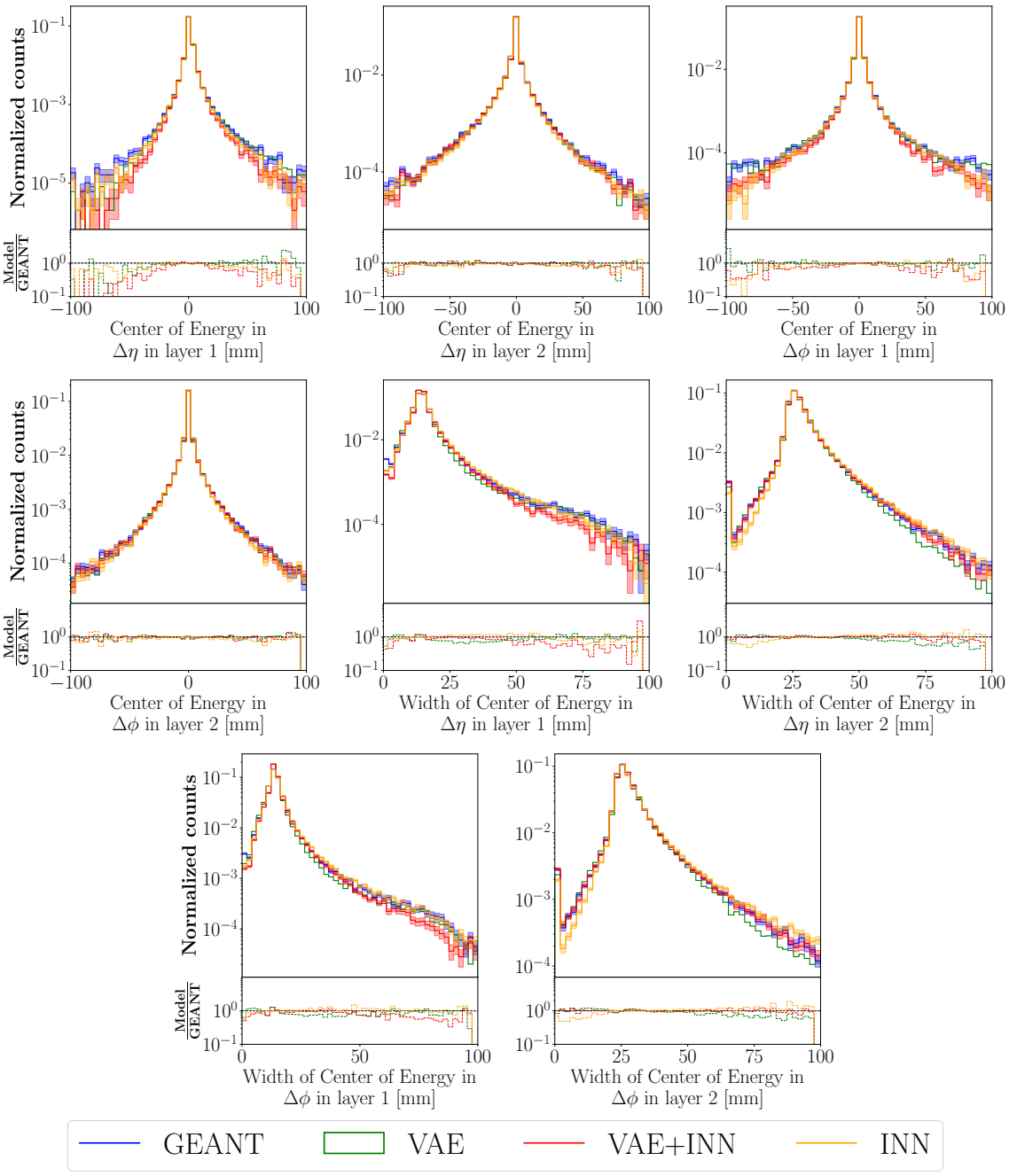

Figure 42: Our final results for the photons in the CaloChallenge dataset 1. In this figure, we show the corresponding shower shape observables.

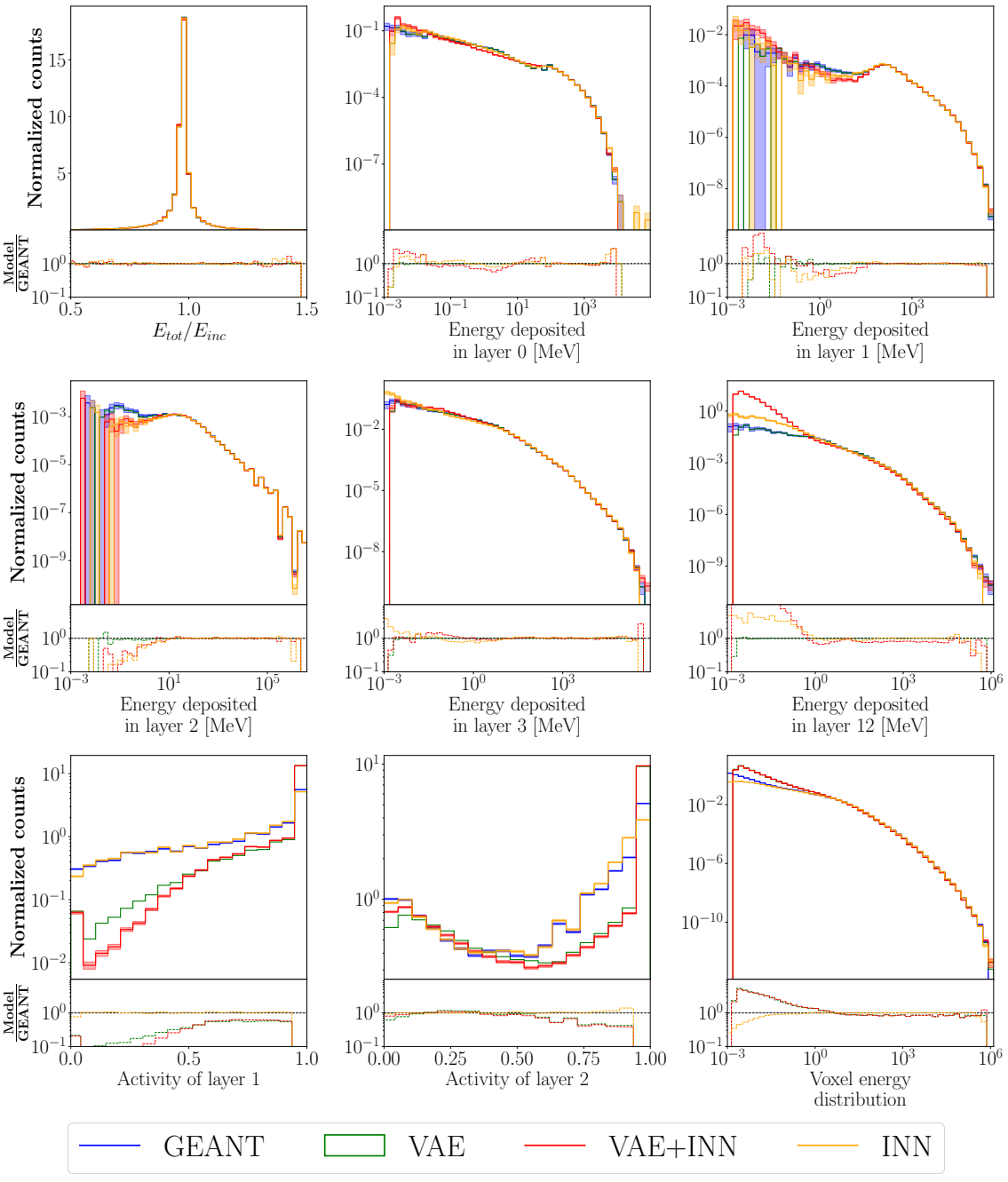

Figure 43: Our final results for the photons in the CaloChallenge dataset 1. In this figure, we show the layer activities and the energy distributions.

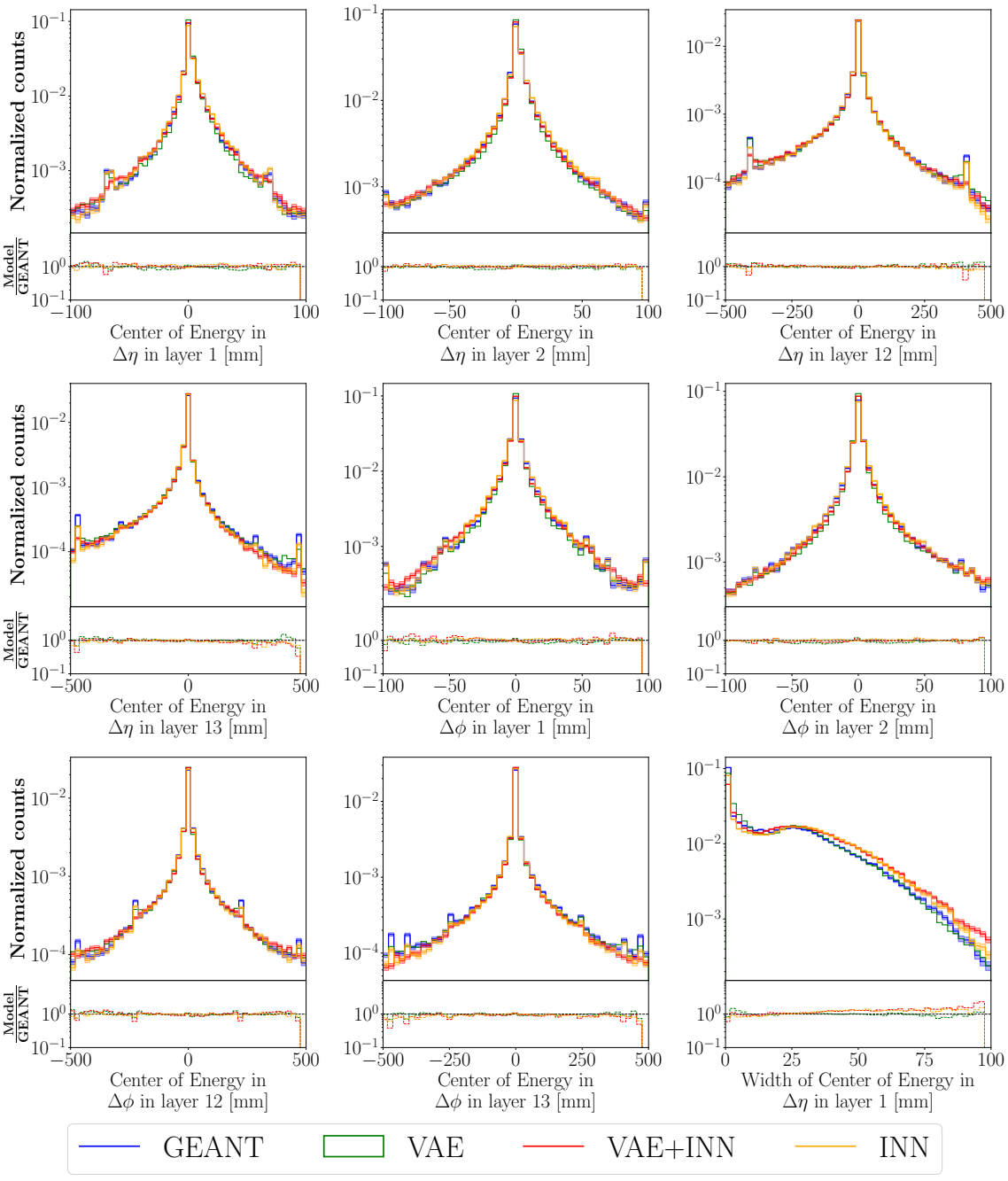

Figure 44: Our final results for the pions in the CaloChallenge dataset 1. In this figure, we show the corresponding shower shape observables.

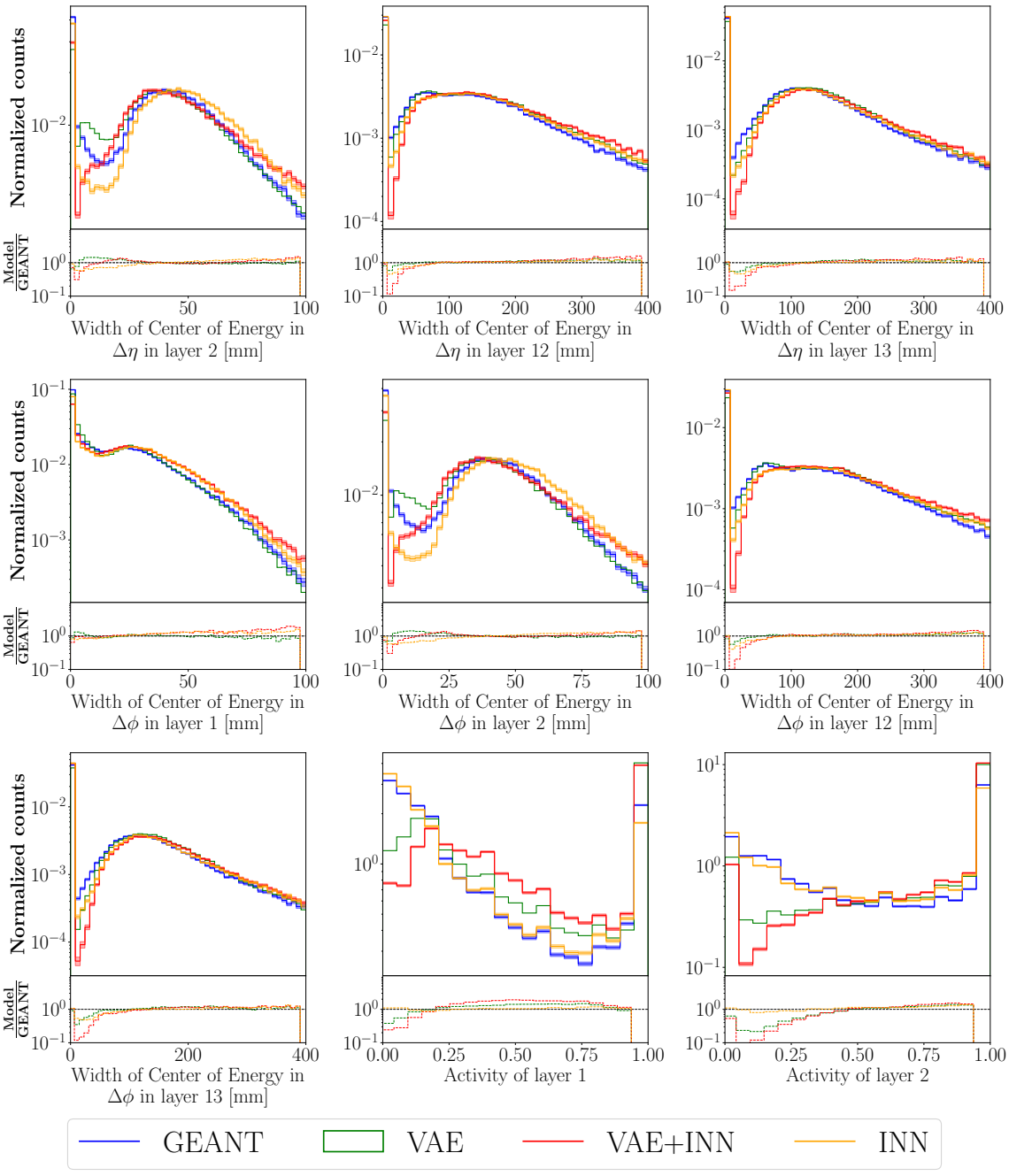

Figure 45: Our final results for the pions in the CaloChallenge dataset 1. In this figure, we show the corresponding shower shape observables and the layer activities.

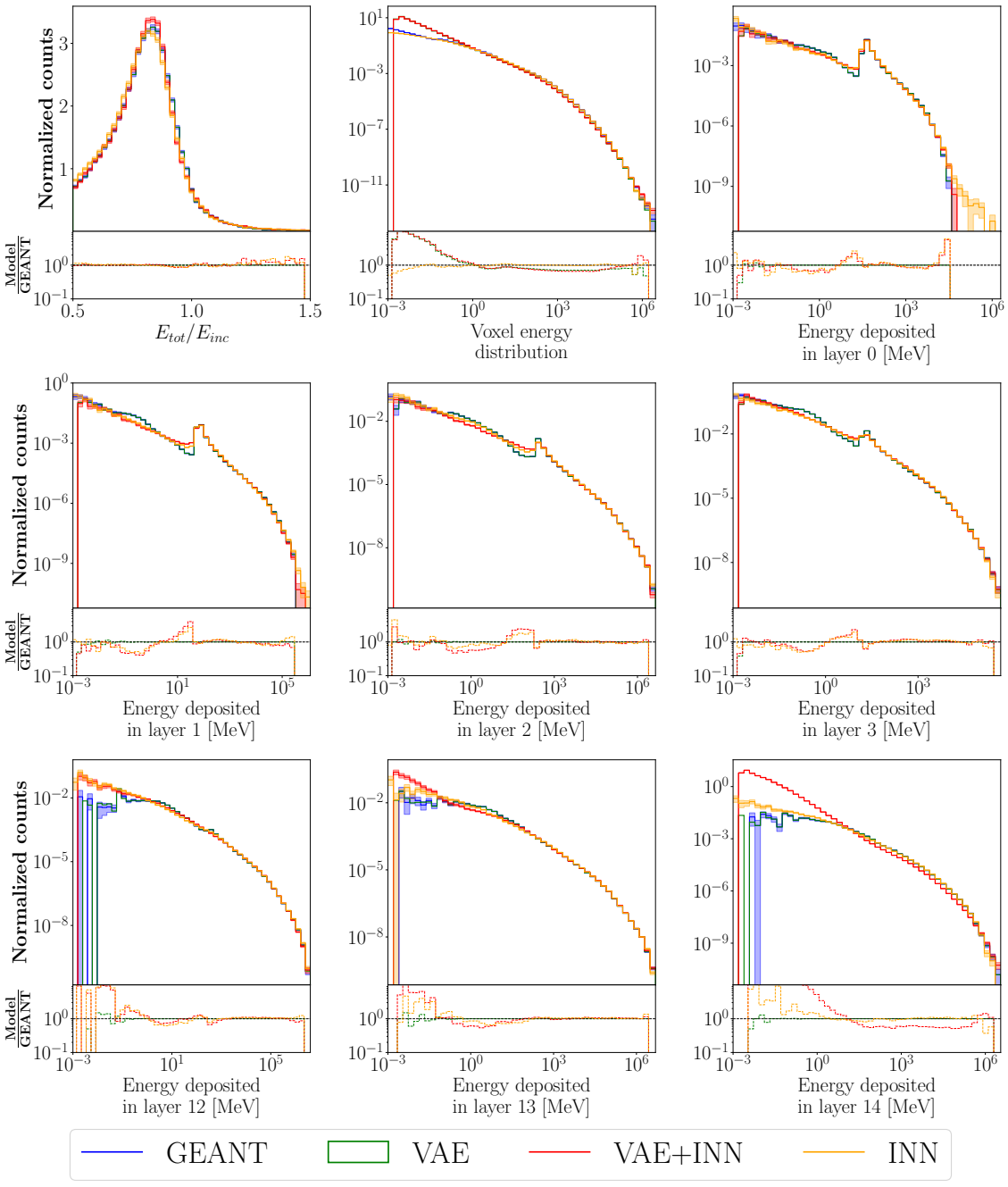

Figure 46: Our final results for the pions in the CaloChallenge dataset 1. In this figure, we show the corresponding energy distributions.

4.4.2 CaloChallenge dataset 2

Moving on to dataset 2 of the CaloChallenge, we realized that our current approach was not practical for large input dimensions. Since we used a fully connected architecture, the number of our model's parameters scaled primarily with the product of input dimensionality n_{input} and the neurons in the first layer n_0:

$$\#\text{parameters} = O(n_{\text{input}} \cdot n_0) = O(n_{\text{input}}^2)$$

The last equality holds as we are scaling our input layer such that it is five to ten times the input dimensionality.
Nevertheless, we tried to apply this setup directly to the CaloChallenge dataset 2. In order to store the model on the GPU memory, we had to reduce the first layer's size significantly below the input dimensionality. In fact, the constraint to keep the same network architecture as before resulted in 5000 neurons in the first layer. Furthermore, we tried to use the pure BCE loss and the setup with additional activity and logit loss. The high level observables were better for the additional loss terms (Figure 47), but the classifier AUC was 0.99. Since the AUC of the pure BCE setup was 0.97, we were assuming that the additional losses had to introduce some artifacts that were invisible in the plots. As both results are, as expected, not acceptable, we do not investigate the classifier weights here further.

To improve our scaling behavior, we developed a different architecture that we call the kernel-VAE, short KVAE. It is a mixture of a fully connected setup and a convolutional one. The main assumption that we made is that the inflation of our first layer is not needed relative to the full input, but to the size of a single layer. We think that the inflation in the first layer enables the model to describe highly correlated features better, for example the activity. However, these correlations should only be strong in a local area. For example we expect the first layer and the last layer to not be strongly correlated.
Building on this assumption, and the observation that the reconstruction was working for fewer layers before, we decided to split the problem into parts: Every l layers are compressed individually in a fully connected sub-network. The first layer of each block has a distance of s from the previous one. The parameters l and s were inspired by the kernel size and the stride of a convolutional network. However, for our setup we learn each of these kernels, instead of shifting the same kernel over the input space.
In the next step the overlapping compressed layer groups are gathered by an inner network, a usual VAE.
After the decoding of the inner VAE, we use the kernels again, to upscale the data to the original data space. We handle the overlapping regions by summing over all predictions for each layer. A visualization of the KVAE architecture can be seen in Figure 49.
The application of the KVAE was greatly improving the quality of our reconstructions as one can see in Figure 47. We chose $l = 7$ and $s = 3$ to have at least two kernels seeing every layer. We also tried different variations for the kernel size and the stride, but did not experience huge differences between them, as long as the stride was not to large with respect to the size. Furthermore, we found the pure BCE loss to be better than the additional losses in all our metrics, when using the KVAE.

A second important improvement was found by coincidence. Because of a bug in the preprocessing step that scales and shifts the data to zero mean and unit variance, we realized that this fixed norm layer was not optimal. When we added an element wise learnable linear layer in the preprocessing and its analytical inverse in the postprocessing, we were able to improve the reconstructions further. Interestingly, this was only helping for our results with the KVAE, meaning dataset 2 (and dataset 3). For dataset 1 and the normal VAE we found it inferior to the fixed standardization.

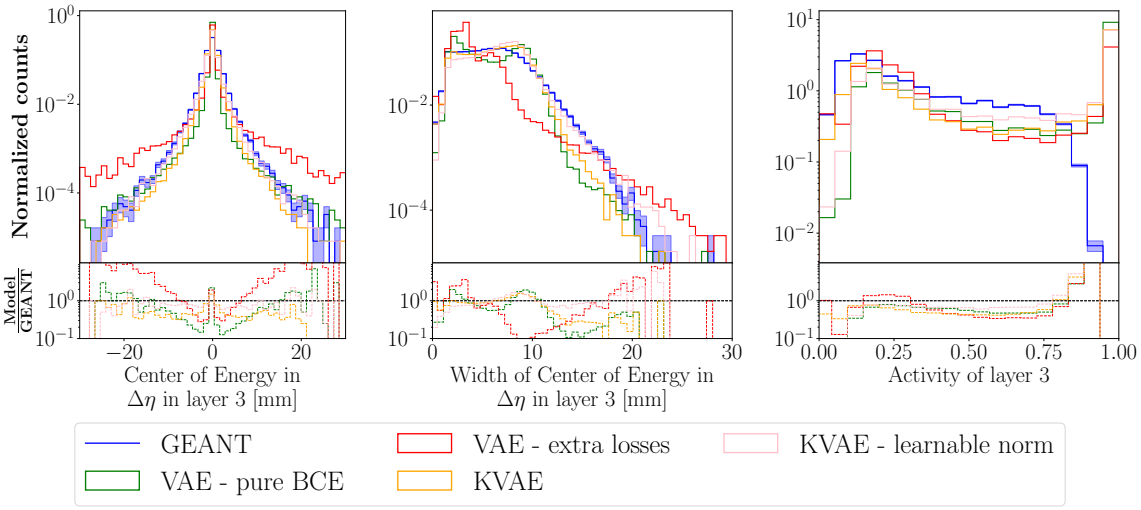

Figure 47: Improvements for dataset 2. The red and green line represent the "normal" VAE results. The KVAE reconstructions are superior in almost every aspect. Only the extra losses setup is slightly superior for the activities. By making our standardization layer learnable we were able to improve the results even further. The pink line is closer to the ground truth than the yellow line for all shower shape related observables.

The corresponding final histograms for dataset 2, using the KVAE with the learnable norm, can be seen in Figure 50, Figure 51 and Figure 52. The results of a classifier weight analysis are given in Figure 48. Again, the pure INN results are much better. However, the gap between the VAE+INN samples and the pure INN samples is closing, implying that the VAE is better in working with high dimensional datasets. When looking at the electromagnetic shower generations of dataset 1 (photons) and dataset 2 (positrons), we see that the VAE based generation results in a "better" AUC, while the pure INN generation results in a worse "AUC". Also the histograms seem to be more similar. Only the layer activity is still a major issue. A full list of all hyperparameters can be seen in Table 5

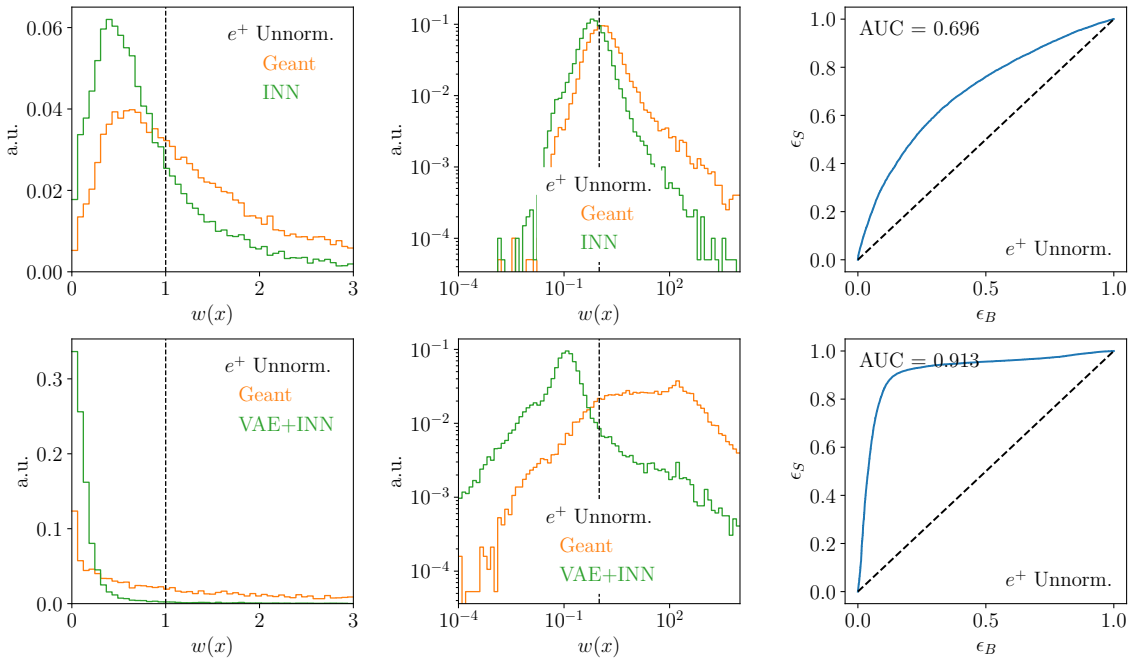

Figure 48: Classifier trained on dataset 2 showers.
(top) The results for the pure INN.
(bottom) The results for the combination of VAE and INN.
Weights distribution in linear space (left) and in log-space (center). (right) ROC curve and relative AUC score.

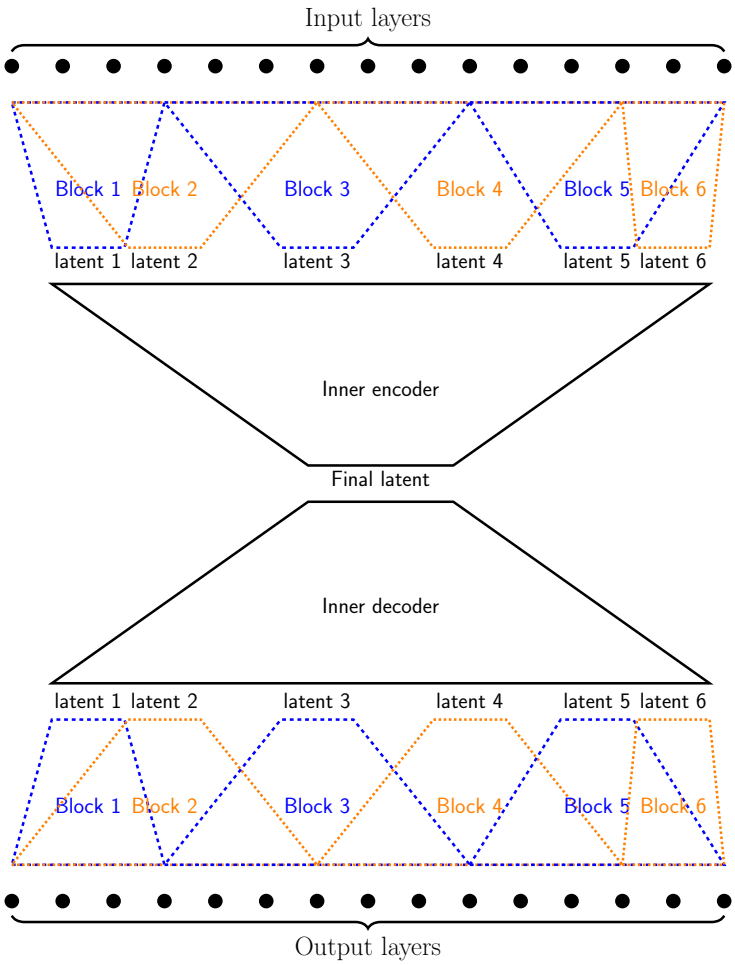

Figure 49: Schematic visualization of the KVAE architecture

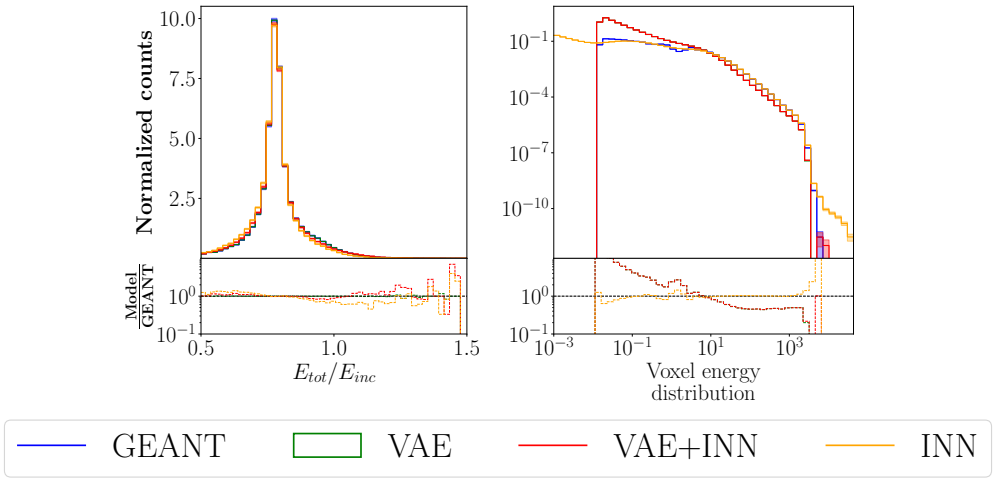

Figure 50: Our final results for the CaloChallenge dataset 2. In this figure, we show the voxel energy distribution and the ratio between E_{tot} and E_{inc}

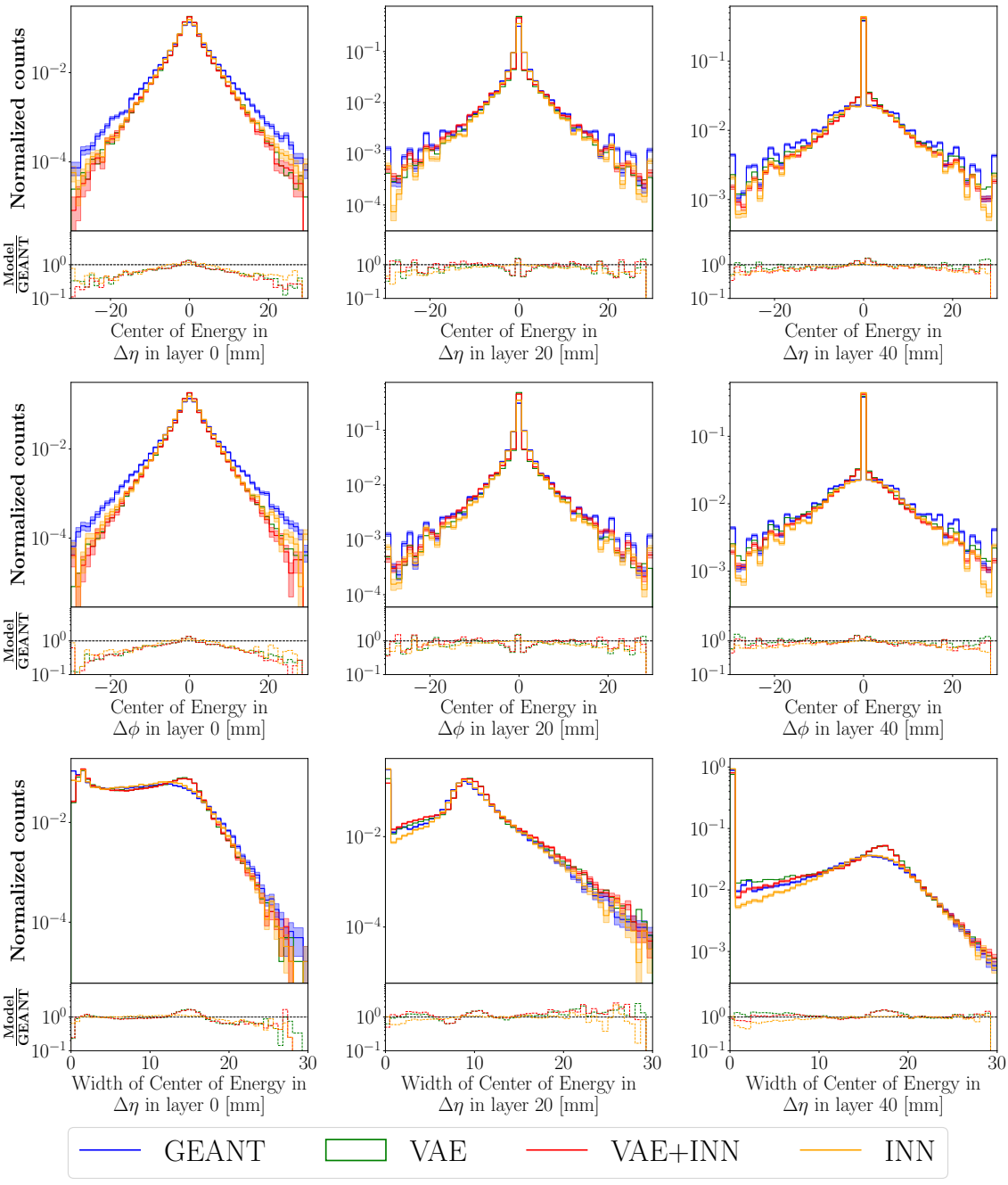

Figure 51: Our final results for the CaloChallenge dataset 2. In this figure, we show some shower shape observables.

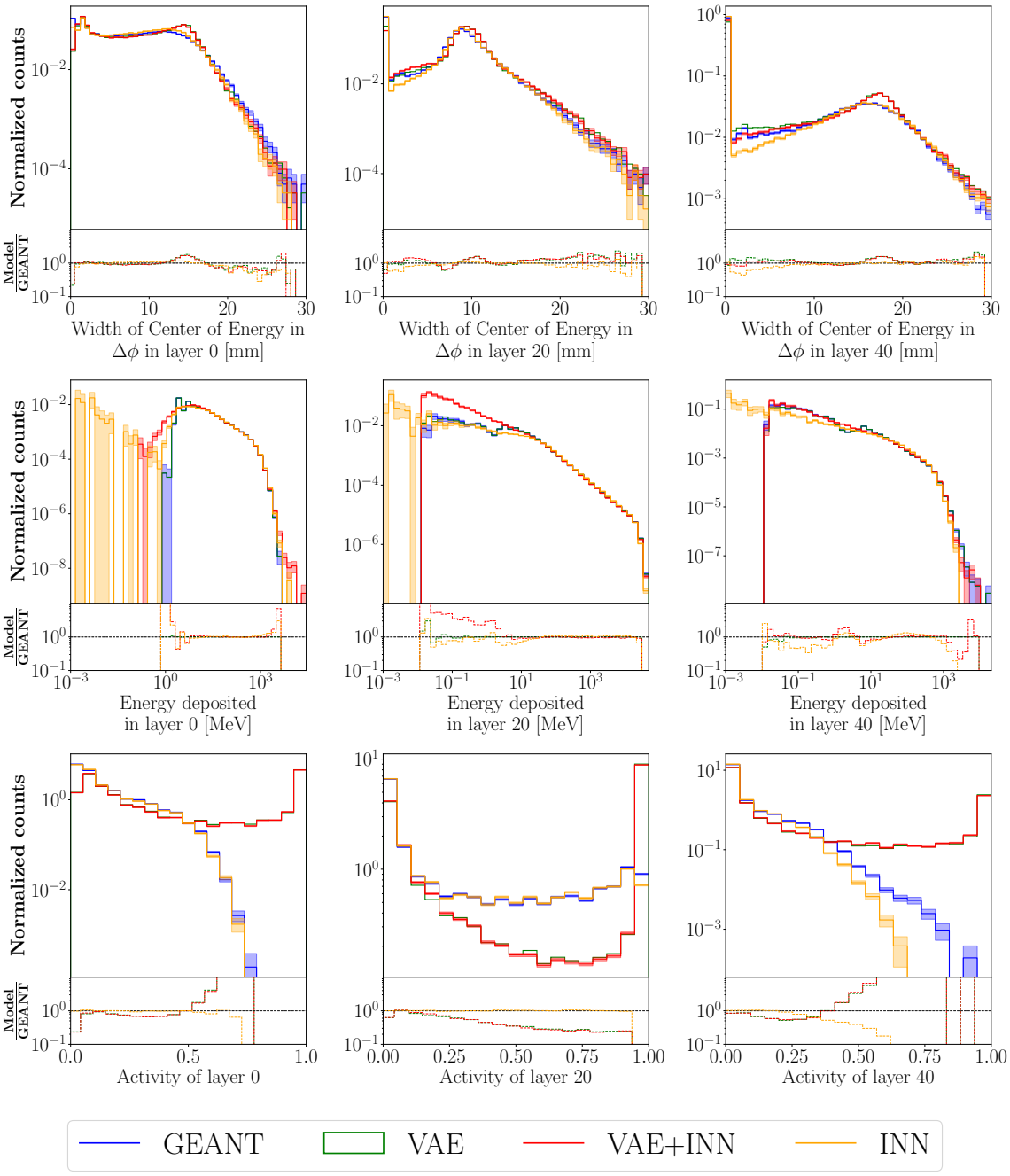

Figure 52: Our final results for the CaloChallenge dataset 2. In this figure, we show further shower shape observables, some activities and some layer energy distributions.

4.4.3 CaloChallenge dataset 3

The third dataset of the CaloChallenge was mainly a challenge because of its sheer size. However, after some memory optimizations of our training routine and an increase of the stride from $s = 3$ to $s = 5$, we were able to run the KVAE setup also for this dataset. Additionally, it turned out to be beneficial to increase all the latent spaces. Further parameter optimizations were not possible due to computational resources. We mainly use the configuration from dataset 2. The final high level observables can be seen in Figure 54, Figure 55 and Figure 56 and the classifier weights are depicted in Figure 53. The classifier results are better than for dataset 2. However, we believe that this is only the case since the classifier with 512 neurons was not expressive enough. However, this architecture was defined by the CaloChallenge and we did not want to modify it too much. This time, no INN comparison can be given, as the INN was not trainable due to the size of dataset 3. All hyperparameters are listed in Table 5.

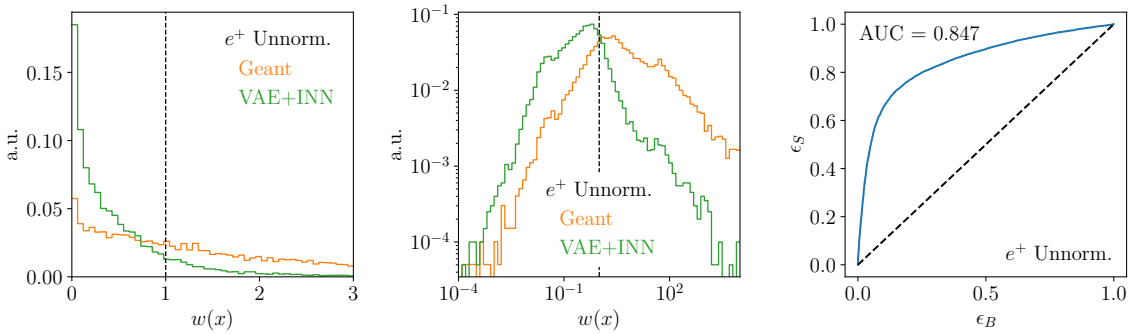

Figure 53: Classifier trained on dataset 3 showers, generated by our VAE+INN architecture. Weights distribution in linear space (left) and in log-space (center). (right) ROC curve and relative AUC score.

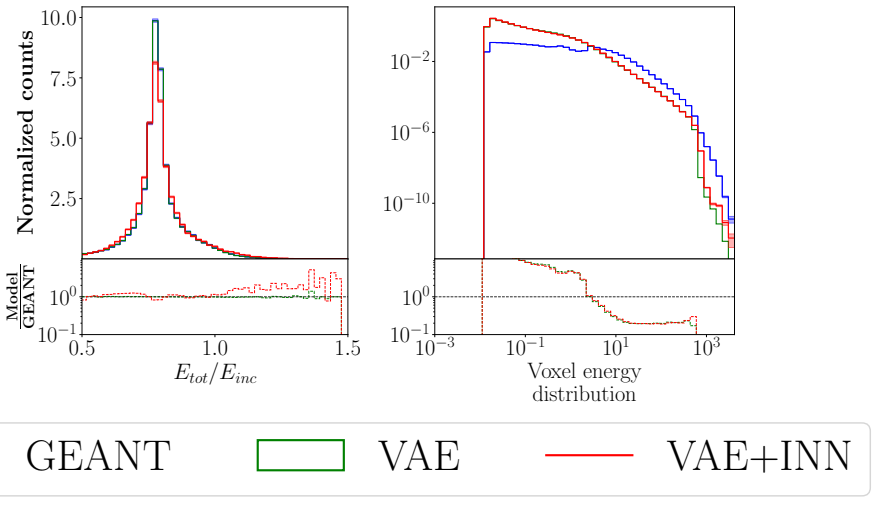

Figure 54: Our final results for the CaloChallenge dataset 3. In this figure, we show the voxel energy distribution and the ratio between E_{tot} and E_{inc}

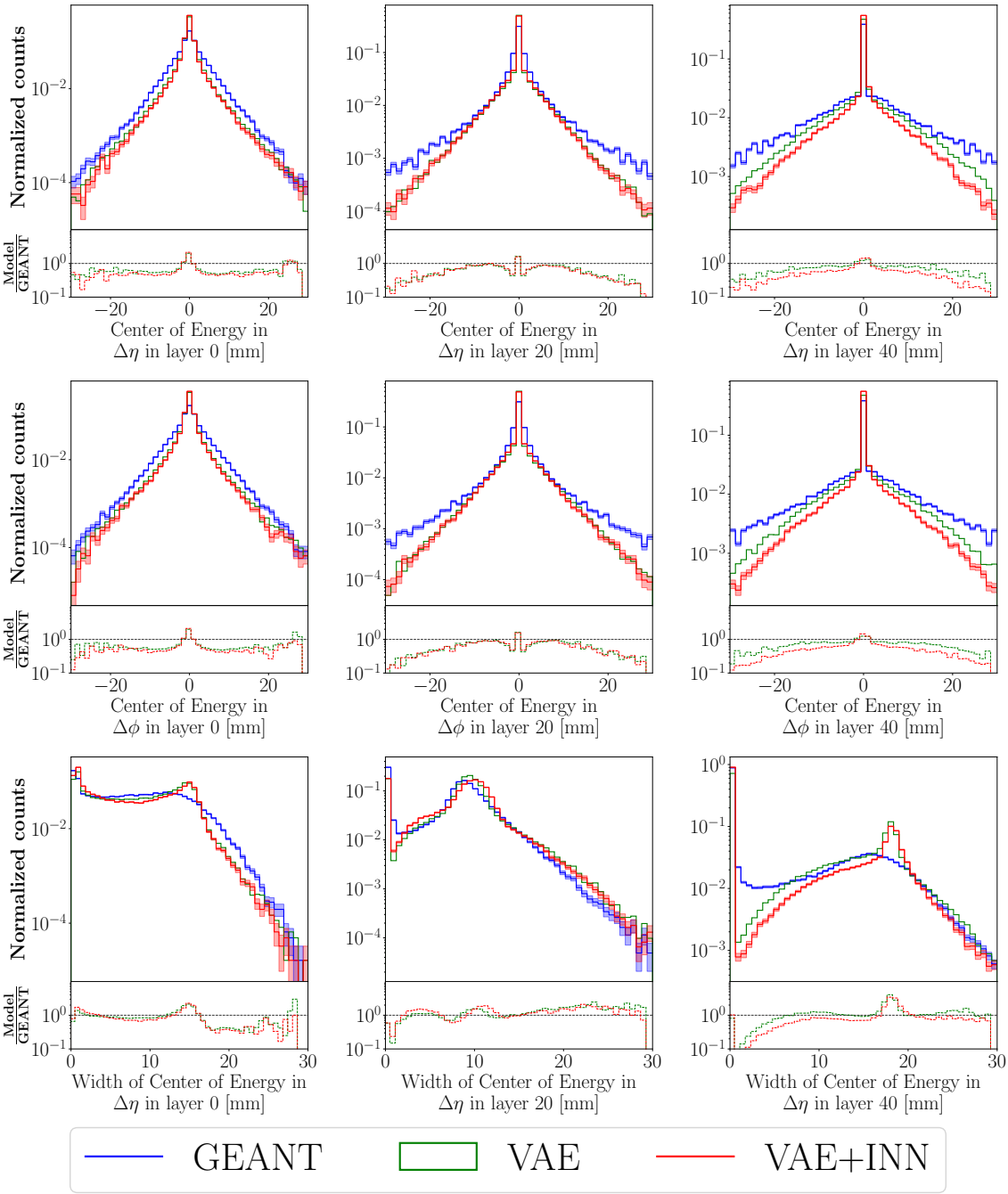

Figure 55: Our final results for the CaloChallenge dataset 3. In this figure, we show some shower shape observables.

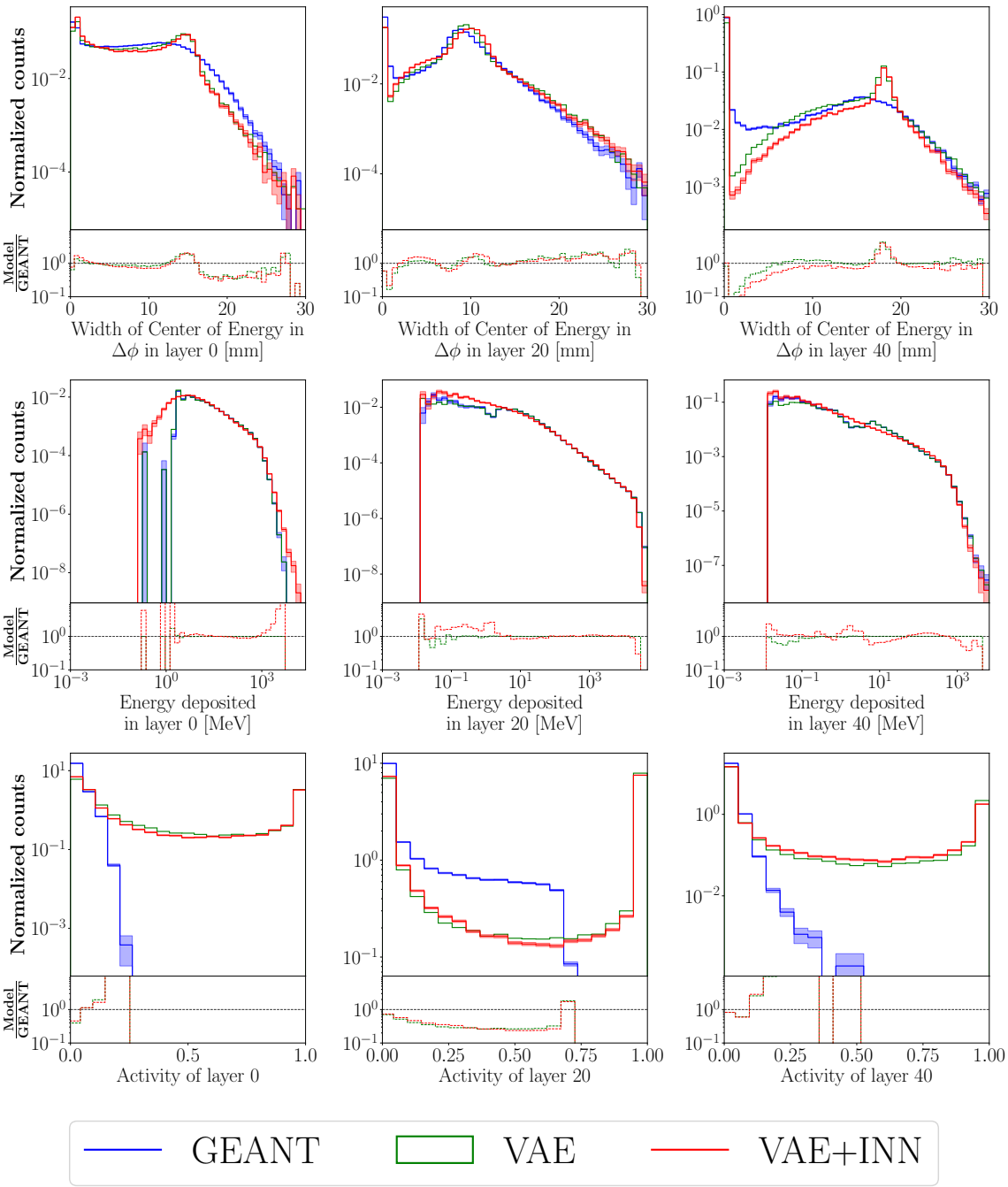

Figure 56: Our final results for the CaloChallenge dataset 3. In this figure, we show further shower shape observables, some activities and some layer energy distributions.

5 Conclusion

We investigated different possibilities to enhance detector simulations in terms of speed. This is necessary as we expect the GEANT based calorimeter simulations to be a major bottleneck in future collider runs. Our focus was on invertible neural networks and variational autoencoders. The exact approach was to use the VAE as a flexible preprocessing step that reduces the dimensionality of the data for the accurate INN. However, we found the VAE to be a major limiting factor in the overall generation quality of the VAE+INN setup and investigated several possibilities to improve accuracy of the VAE. Our first major improvement was the addition of a physics motivated "smearing matrix" before the loss function, to enhance the importance of the spatial neighborhood in a layer. Afterwards we were replacing the Gaussian VAE with a Bernoulli VAE, improving the reconstruction quality even further. However, we found that the sensitivity of the BCE loss towards low energies was so good, that the smearing matrix was resulting in a negligible effect as it was weakening the low energy sensitivity. For the larger datasets 2 and 3 were were developing a new model architecture, the KVAE, that is able to handle the large input dimensionality by using a locality assumption. With the KVAE we were able to improve our results significantly, for a third time.
However, after all this improvements, the (K)VAE's reconstructions are still inferior to the pure INN generations, making the VAE the bottleneck in the VAE+INN approach. For hadronic showers, we were able to receive similar weight distributions and high level observables for the VAE+INN and the pure INN setup, for electromagnetic showers, the VAE was not able to reach the level of an INN. So, as long as the INN is applicable, it is reasonable to use it on its own. If the INN is not applicable, due to the size of the input, adding VAE can be a viable alternative for hadronic showers. For electromagnetic showers, however, it might be reasonable to use other approaches like diffusion models. By the end of the CaloChallenge we will learn more about the different alternatives to INNs and VAEs and it will be interesting to see the corresponding advantages and disadvantages.

6 Next steps

If it turns out that the VAE+INN approach is competitive against the other state of the art models, after the CaloChallenge, the next step is to add uncertainty estimates. It is not as easy as for the INN, where we could just the Bayesian NN loss. In the Bayesian NN loss, we are using the likelihood of our dataset given our model, a value that we know for the INN, but not for the VAE. The VAE is assuming a joint probability $p(x, z)$, and trying to use the corresponding conditional probabilities to find a probabilistic mapping between latent space and data space. The INN, on the other hand, is trying to find a transformation of the variable x itself, such that the probability distribution after this variable change is equal to the latent distribution. So, the INN is learning a deterministic reparameterization of our random variable. This difference has the consequence that the INN learns the likelihood of the dataset explicitly but the VAE does not. Therefore, it is not straightforward to turn the VAE into a Bayesian version of itself. However, as the formalism of the VAE and the

Bayesian NN, variational inference, is similar, it might be possible to combine these two approaches. In any case further research would be needed in this area.
Alternatively to the VAE+INN approach, one could use a fully convolutional approach for the INN as it would solve its scaling problems as well. However, this will be difficult as well as we saw that the convolutional results were inferior to the fully connected results, so far. Especially, single active voxels are difficult to generate with convolutional networks, so a solution to prevent bad activities and blurry reconstructions is needed for this setup as well.

References

[1] Michael E. Peskin and Daniel V. Schroeder. *An Introduction to quantum field theory*. Addison-Wesley, Reading, USA, 1995.

[2] M. Srednicki. *Quantum field theory*. Cambridge University Press, 1 2007.

[3] Steven Weinberg. *The Quantum theory of fields. Vol. 1: Foundations*. Cambridge University Press, 6 2005. doi:10.1017/CBO9781139644167.

[4] Sheldon L. Glashow. Partial-symmetries of weak interactions. *Nuclear Physics*, 22(4):579–588, 1961. URL: https://www.sciencedirect.com/science/article/pii/0029558261904692, doi:https://doi.org/10.1016/0029-5582(61)90469-2.

[5] Steven Weinberg. A model of leptons. *Phys. Rev. Lett.*, 19:1264–1266, Nov 1967. URL: https://link.aps.org/doi/10.1103/PhysRevLett.19.1264, doi:10.1103/PhysRevLett.19.1264.

[6] M. Veltman. Perturbation theory of massive yang-mills fields. *Nuclear Physics B*, 7(5):637–650, 1968. URL: https://www.sciencedirect.com/science/article/pii/0550321368901971, doi:https://doi.org/10.1016/0550-3213(68)90197-1.

[7] G.'t Hooft. Renormalizable lagrangians for massive yang-mills fields. *Nuclear Physics B*, 35(1):167–188, 1971. URL: https://www.sciencedirect.com/science/article/pii/0550321371901398, doi:https://doi.org/10.1016/0550-3213(71)90139-8.

[8] David J. Gross and Frank Wilczek. Ultraviolet behavior of non-abelian gauge theories. *Phys. Rev. Lett.*, 30:1343–1346, Jun 1973. URL: https://link.aps.org/doi/10.1103/PhysRevLett.30.1343, doi:10.1103/PhysRevLett.30.1343.

[9] David J. Gross and Frank Wilczek. Asymptotically free gauge theories. i. *Phys. Rev. D*, 8:3633–3652, Nov 1973. URL: https://link.aps.org/doi/10.1103/PhysRevD.8.3633, doi:10.1103/PhysRevD.8.3633.

[10] H. David Politzer. Reliable perturbative results for strong interactions? *Phys. Rev. Lett.*, 30:1346–1349, Jun 1973. URL: https://link.aps.org/doi/10.1103/PhysRevLett.30.1346, doi:10.1103/PhysRevLett.30.1346.

[11] Tilman Plehn. *Lectures on LHC Physics*. Springer Berlin Heidelberg, 2012. URL: https://doi.org/10.1007%2F978-3-642-24040-9, doi:10.1007/978-3-642-24040-9.

[12] Mark Thomson. *Modern particle physics*. Cambridge University Press, New York, 2013. doi:10.1017/CBO9781139525367.

[13] By Cush - Own work using:PBS NOVA, Fermilab, Office of Science, United States Department of Energy, Particle Data Group, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4286964.

[14] C. Amsler et al. Neutrino mass, mixing, and flavor change. *Physics Letters B*, 667(1):163–172, 2008. Review of Particle Physics. URL: https://www.sciencedirect.com/science/article/pii/S0370269308008526, doi:https://doi.org/10.1016/j.physletb.2008.07.027.

[15] F. Englert and R. Brout. Broken Symmetry and the Mass of Gauge Vector Mesons. *prl*, 13(9):321–323, August 1964. doi:10.1103/PhysRevLett.13.321.

[16] Peter W. Higgs. Broken symmetries and the masses of gauge bosons. *Phys. Rev. Lett.*, 13:508–509, Oct 1964. URL: https://link.aps.org/doi/10.1103/PhysRevLett.13.508, doi:10.1103/PhysRevLett.13.508.

[17] G. S. Guralnik, C. R. Hagen, and T. W. B. Kibble. Global conservation laws and massless particles. *Phys. Rev. Lett.*, 13:585–587, Nov 1964. URL: https://link.aps.org/doi/10.1103/PhysRevLett.13.585, doi:10.1103/PhysRevLett.13.585.

[18] G. Aad et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett. B*, 716:1–29, 2012. arXiv:1207.7214, doi:10.1016/j.physletb.2012.08.020.

[19] H. Bethe. Zur theorie des durchgangs schneller korpuskularstrahlen durch materie. *Annalen der Physik*, 397(3):325–400, 1930. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19303970303, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.19303970303, doi:https://doi.org/10.1002/andp.19303970303.

[20] G. Aad et al. AtlFast3: The Next Generation of Fast Simulation in ATLAS. *Comput. Softw. Big Sci.*, 6(1):7, 2022. arXiv:2109.02551, doi:10.1007/s41781-021-00079-7.

[21] Frank Hartmann and Archana Sharma. Multipurpose detectors for high energy physics, an introduction. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 666:1–9, 2012. Advanced Instrumentation. URL: https://www.sciencedirect.com/science/article/pii/S0168900211020626, doi:https://doi.org/10.1016/j.nima.2011.11.023.

[22] https://cds.cern.ch/record/1505342. Last accessed on 22.10.2023. CERN is the source of the image.

[23] Frank Hartmann. Silicon tracking detectors in high-energy physics. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 666:25–46, 2012. Advanced Instrumentation. URL: https://www.sciencedirect.com/science/article/pii/S0168900211020389, doi:https://doi.org/10.1016/j.nima.2011.11.005.

[24] R.M. Brown and D.J.A. Cockerill. Electromagnetic calorimetry. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 666:47–79, 2012. Advanced Instrumentation. URL: https://www.sciencedirect.com/science/article/pii/S0168900211005572, doi:https://doi.org/10.1016/j.nima.2011.03.017.

[25] Nural Akchurin and Richard Wigmans. Hadron calorimetry. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 666:80–97, 2012. Advanced Instrumentation. URL: https://www.sciencedirect.com/science/article/pii/S0168900211019851, doi:https://doi.org/10.1016/j.nima.2011.10.035.

[26] S. Agostinelli et al. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003. URL: https://www.sciencedirect.com/science/article/pii/S0168900203013688, doi:https://doi.org/10.1016/S0168-9002(03)01368-8.

[27] J. Allison et al. Geant4 developments and applications. *IEEE Transactions on Nuclear Science*, 53(1):270–278, 2006. doi:10.1109/TNS.2006.869826.

[28] J Allison et al. Recent developments in geant4. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 835:186–225, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0168900216306957, doi:https://doi.org/10.1016/j.nima.2016.06.125.

[29] ATLAS HL-LHC Computing Conceptual Design Report. 2020.

[30] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters. *Phys. Rev. Lett.*, 120(4):042003, 2018. arXiv:1705.02355, doi:10.1103/PhysRevLett.120.042003.

[31] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Phys. Rev. D*, 97(1):014021, 2018. arXiv:1712.10321, doi:10.1103/PhysRevD.97.014021.

[32] Martin Erdmann, Lukas Geiger, Jonas Glombitza, and David Schmidt. Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks. *Comput. Softw. Big Sci.*, 2(1):4, 2018. arXiv:1802.03325, doi:10.1007/s41781-018-0008-x.

[33] Martin Erdmann, Jonas Glombitza, and Thorben Quast. Precise simulation of electromagnetic calorimeter showers using a wasserstein generative adversarial network. *Computing and Software for Big Science*, 3(1), jan 2019. URL: https://doi.org/10.1007%2Fs41781-018-0019-7, doi:10.1007/s41781-018-0019-7.

[34] Deep generative models for fast shower simulation in ATLAS. Technical report, CERN, Geneva, 2018. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-SOFT-PUB-2018-001. URL: https://cds.cern.ch/record/2630433.

[35] Dawit Belayneh et al. Calorimetry with deep learning: particle simulation and reconstruction for collider physics. *The European Physical Journal C*, 80(7), jul 2020. URL: https://doi.org/10.1140%2Fepjc%2Fs10052-020-8251-9, doi:10.1140/epjc/s10052-020-8251-9.

[36] Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, and Katja Krüger. Getting high: High fidelity simulation of high granularity calorimeters with high speed. *Computing and Software for Big Science*, 5(1), may 2021. URL: https://doi.org/10.1007%2Fs41781-021-00056-0, doi:10.1007/s41781-021-00056-0.

[37] Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, and Katja Krüger. Decoding photons: Physics in the latent space of a BIB-AE generative network. *EPJ Web of Conferences*, 251:03003, 2021. URL: https://doi.org/10.1051%2Fepjconf%2F202125103003, doi:10.1051/epjconf/202125103003.

[38] Fast simulation of the ATLAS calorimeter system with Generative Adversarial Networks. Technical report, CERN, Geneva, 2020. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-SOFT-PUB-2020-006. URL: https://cds.cern.ch/record/2746032.

[39] Claudius Krause and David Shih. CaloFlow: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows. 6 2021. arXiv:2106.05285.

[40] Claudius Krause and David Shih. CaloFlow II: Even Faster and Still Accurate Generation of Calorimeter Showers with Normalizing Flows. 10 2021. arXiv:2110.11377.

[41] Claudius Krause, David Shih, and Dalila Salamani. Fast calorimeter simulation challenge. https://github.com/CaloChallenge/homepage, 2023.

[42] Tilman Plehn, Anja Butter, Barry Dillon, and Claudius Krause. Modern machine learning for lhc physicists, 2022. arXiv:2211.01421.

[43] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Qasim Al-Dujaili, Ye Duan, Omran Al-Shamma, José I. Santamaría, Mohammed Abdulraheem Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8, 2021. URL: https://api.semanticscholar.org/CorpusID:232434552.

[44] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018. arXiv:1708.07120.

[45] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL: http://arxiv.org/abs/1412.6980.

[46] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[47] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL: https://api.semanticscholar.org/CorpusID:205001834.

[48] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[49] Alberto Prieto, Beatriz Prieto, Eva Martinez Ortigosa, Eduardo Ros, Francisco Pelayo, Julio Ortega, and Ignacio Rojas. Neural networks: An overview of early research, current frameworks and new challenges. *Neurocomputing*, 214:242–268, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0925231216305550, doi:https://doi.org/10.1016/j.neucom.2016.06.014.

[50] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. URL: https://www.sciencedirect.com/science/article/pii/S0893608014002135, doi:https://doi.org/10.1016/j.neunet.2014.09.003.

[51] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989. URL: https://api.semanticscholar.org/CorpusID:3958369.

[52] Jan Holstermann. On the expressive power of neural networks, 2023. arXiv:2306.00145.

[53] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998. doi:10.1109/5.726791.

[54] J. Neyman and E. S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231:289–337, 1933. URL: http://www.jstor.org/stable/91247.

[55] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests, 2018. arXiv:1610.06545.

[56] Ranit Das, Luigi Favaro, Theo Heimel, Claudius Krause, Tilman Plehn, and David Shih. How to understand limitations of generative networks, 2023. arXiv:2305.16774.

[57] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation, 2015. arXiv:1410.8516.

[58] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, nov 2021. URL: https://doi.org/10.1109%2Ftpami.2020.2992934, doi:10.1109/tpami.2020.2992934.

[59] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference, 2021. arXiv:1912.02762.

[60] Ramon Winterhalder, Marco Bellagente, and Benjamin Nachman. Latent space refinement for deep generative models, 2021. arXiv:2106.00792.

[61] https://hci.iwr.uni-heidelberg.de/vislearn/inverse-problems-invertible-neural-networks. Last accessed on 22.10.2023.

[62] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling, 2019. arXiv:1808.03856.

[63] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows, 2019. arXiv:1906.04032.

[64] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows, 2019. arXiv:1906.02145.

[65] M. Steffen. A simple method for monotonic interpolation in one dimension. *aap*, 239:443–450, November 1990.

[66] J. A. GREGORY and R. DELBOURGO. Piecewise Rational Quadratic Interpolation to Monotonic Data. *IMA Journal of Numerical Analysis*, 2(2):123–130, 04 1982. arXiv:https://academic.oup.com/imajna/article-pdf/2/2/123/2267745/2-2-123.pdf, doi:10.1093/imanum/2.2.123.

[67] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks, 2019. arXiv:1907.02392.

[68] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation, 2018. arXiv:1705.07057.

[69] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow, 2017. arXiv:1606.04934.

[70] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks, 2018. arXiv:1802.07088.

[71] Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks, 2019. arXiv:1811.00995.

[72] Lynton Ardizzone, Till Bungert, Felix Draxler, Ullrich Köthe, Jakob Kruse, Robert Schmier, and Peter Sorrenson. Framework for Easily Invertible Architectures (FrEIA), 2018-2022. URL: https://github.com/vislearn/FrEIA.

[73] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick, 2015. arXiv:1506.02557.

[74] Gregor Kasieczka, Michel Luchmann, Florian Otterpohl, and Tilman Plehn. Per-object systematics using deep-learned calibration. *SciPost Physics*, 9(6), dec 2020. URL: https://doi.org/10.21468%2Fscipostphys.9.6.089, doi:10.21468/scipostphys.9.6.089.

[75] Marco Bellagente, Manuel Haussmann, Michel Luchmann, and Tilman Plehn. Understanding event-generation networks via uncertainties. *SciPost Physics*, 13(1), jul 2022. URL: https://doi.org/10.21468%2Fscipostphys.13.1.003, doi:10.21468/scipostphys.13.1.003.

[76] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. arXiv:1312.6114.

[77] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL: https://openreview.net/forum?id=Sy2fzU9gl.

[78] Gabriel Loaiza-Ganem and John P. Cunningham. The continuous bernoulli: fixing a pervasive error in variational autoencoders, 2019. arXiv:1907.06845.

[79] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[80] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[81] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. arXiv:1502.03167.

[82] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008. doi:10.1088/1748-0221/3/08/S08003.

[83] Robin Rombach. Normalizing flows for enhancing and explaining deep neural networks, 2020.

[84] Patrick Esser, Robin Rombach, and Björn Ommer. A disentangling invertible interpretation network for explaining latent representations, 2020. arXiv:2004.13166.

[85] Jesse C. Cresswell, Brendan Leigh Ross, Gabriel Loaiza-Ganem, Humberto Reyes-Gonzalez, Marco Letizia, and Anthony L. Caterini. CaloMan: Fast generation of

calorimeter showers with density estimation on learned manifolds. In *36th Conference on Neural Information Processing Systems*, 11 2022. arXiv:2211.15380.

[86] George Stein et al. A probabilistic autoencoder for type ia supernova spectral time series. *The Astrophysical Journal*, 935(1):5, aug 2022. URL: https://doi.org/10.3847%2F1538-4357%2Fac7c08, doi:10.3847/1538-4357/ac7c08.

[87] Oleh Rybkin, Kostas Daniilidis, and Sergey Levine. Simple and effective vae training with calibrated decoders, 2021. arXiv:2006.13202.

[88] Dalila Salamani. *Machine Learning Techniques for Fast Shower Simulation at the ATLAS Experiment*. PhD thesis, Geneva U., 2021. doi:10.13097/archive-ouverte/unige:158540.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Tilman Plehn for giving me the possibility to write this thesis and his support while doing so. I want to thank him especially for his effort to help me finding a PhD position with a similar topic.
I would also like to thank my collaborators Dr. Claudius Krause and Luigi Favaro for their ongoing support and their constructive feedback during this research project. I want to thank them for the proofreading of this thesis as well, helping me to remove a lot of conceptual and grammatical errors.
Additionally, I appreciated the constructive input from Prof. David Shih. His insights on questioning and improving the VAE architecture were invaluable during the second half of my master's project.
Furthermore, I am also grateful to Prof. Björn Malte Schäfer for kindly agreeing to be my second corrector for this thesis. I appreciate his willingness to undertake the demanding task of reviewing it.
Last, but not least, I want to thank all the people in our group for their help and time, whenever I had any questions or problems.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den **20.11.2023** .....