LUND UNIVERSITY

LTH

FACULTY OF ENGINEERING

Deep Autoencoders for Data Compression in High Energy Physics

Eric Wulff

Thesis submitted for the degree of
Master of Science in Engineering

Degree Project in Physics: PHYM01

Supervisors:
Caterina Doglioni, Antonio Boveia, and Lukas Heinrich

February 9, 2020

Abstract

Current technological limitations make it impossible to store the enormous amount of data produced from proton-proton collisions by the ATLAS detector at CERN's Large Hadron Collider. Therefore, specialised hardware and software is being used to decide what data, or which proton-proton *collision events*, to save and which to discard. A reduction in the storage size of each collision event is desirable as it would allow for more data to be saved and thereby make a larger set of physics analyses possible.

The focus of this thesis is to understand whether it is possible to reduce the storage size of previously mentioned collision events using machine learning techniques for dimensionality reduction. This has never before been tried within the ATLAS experiment and is an interesting forward-looking study for future experiments. Specifically, autoencoder neural networks are used to compress a number of variables into a smaller latent space used for storage. Different neural network architectures with varying width and depth are explored.

The AEs are trained and validated on experimental data from the ATLAS detector and their performance is tested on an independent signal Monte-Carlo sample. The AEs are shown to successfully compress and decompress simple hadron jet data and preliminary results indicate that the reconstruction quality is good enough for certain applications where high precision is not paramount. The AEs are evaluated by their reconstruction error, the relative error of each compressed variable and the ability to retain good resolution of a dijet mass signal (from the previously mentioned Monte-Carlo sample) after encoding and decoding hadron jet data.

Preface

This thesis report was submitted for the degree of Master of Science in Engineering at the Faculty of Engineering (LTH) at Lund University (LU) in January 2020.

Acknowledgments

Thanks to my principal supervisor Caterina Doglioni (LU) as well as to my assistant supervisors Antonio Boveia (OSU) and Lukas Heinrich (CERN) for great support during this thesis project.

Thanks to Ayla Borglund for comments on writing style and language.

The training of deep learning models was performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at LUNARC and the High Performance Computing Center North (HPC2N). Birgitte Brydsoe and Lars Viklund at HPC2N is acknowledged for assistance concerning technical support in making the code run on the HPC2N resources.

Contents

List of Abbreviations

CERN = Conseil Européen pour la Recherche Nucléaire
LHC = Large Hadron Collider
ATLAS = A Toroidal LHC Apparatus
TLA = Trigger Level Analysis
QCD = Quantum Chromodynamics
AI = Artificial Intelligence
ML = Machine Learning
NN = Neural Network
DNN = Deep Neural Network
DL = Deep Learning
AE = Autoencoder

Chapter 1

Introduction

1.1 Background and motivation

1.1.1 CERN and the Large Hadron Collider

The European Organisation for Nuclear Research (CERN) is one of the largest research laboratories in the world and has as its mission to conduct world-class scientific research in fundamental physics as well as to push the frontiers of science and technology. The main scientific tools used at CERN are particle accelerators and detectors. In fact, CERN is home to the largest particle accelerator in the world, the Large Hadron Collider (LHC), perhaps best known for having been used to discover the Higgs boson [1].

The high technological demands of building particle accelerators for use in particle physics research result in CERN pushing the frontiers of technology, often to the benefit of society at large. As an example, the idea for perhaps one of the most important inventions in human history, the World Wide Web, was first imagined at CERN [2]. Medical diagnosis and treatment are other areas where CERN scientists have helped make big technological advances. Medical applications that rely on technology first developed for particle physics research includes PET scans, MRI scans (for diagnosis) and hadron therapy (cancer treatment).

The LHC accelerates protons to 99.999999% the speed of light (that is a mere 10 km/h slower than light) and make them collide at 4 different locations along the accelerator ring. At these locations, particle detectors probe the physics of the proton-proton collisions. Due to Einstein's mass-energy equivalence [3], $E = mc^2$, the energy of the colliding protons can be converted into many different stable and unstable particles, which in turn can decay into other particles before they reach the detectors. Physicists analyse the recorded data in order to infer what happened between the proton-proton collisions and the detection of particles in the detector.

1.1.2 Theory

A brief introduction to a few fundamental concepts in physics necessary for the understanding of this thesis will be given here. Readers already familiar with special relativity, elementary particle physics and the Standard Model (SM) can skip this section.

The Standard Model

The Standard Model (SM) of particle physics is our current best theory of how the universe works. It describes the so-called *fundamental particles* and how they interact with each other. All particles included in the SM is shown in Fig. 1.1. There, it can also be seen that there are different families of particles, such as *fermions* and *bosons* or *quarks* and *leptons*. Most of the matter we see in our every day lives consist of just 3 of these fundamental particles, the up and the down quarks, which combine together to create protons and neutrons, and the electron.

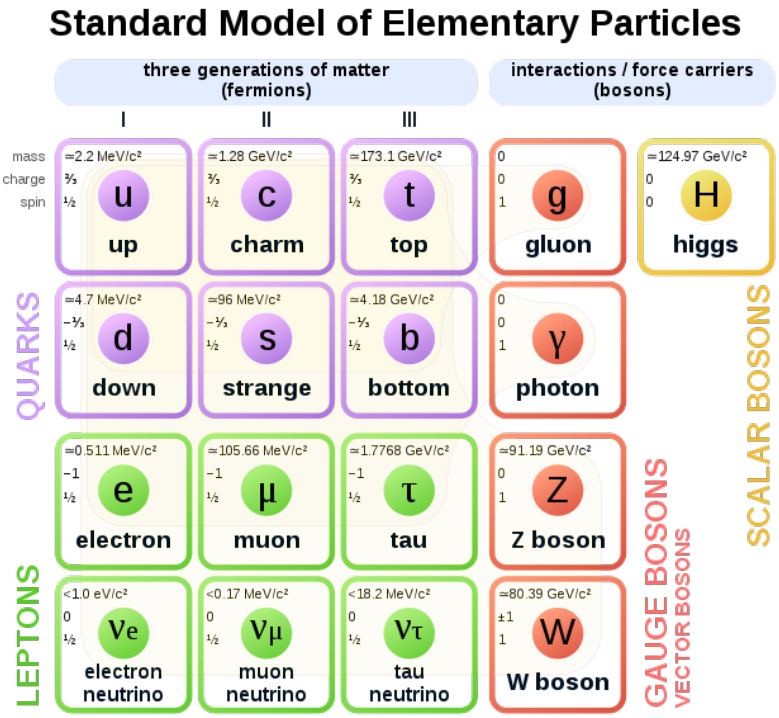

Figure 1.1: The fundamental particles of the Standard Model of particle physics. [4]

4-momentum

4-momentum is a relativistic generalization of the 3-dimensional momentum in classical mechanics. It has the practical property that its magnitude is invariant under Lorentz transformations, or in other words, its magnitude is the same in all inertial reference frames. As suggested by its name, 4-momentum has 4 elements and is defined as

$$p^\mu = (E, p_x, p_y, p_z) \ , \tag{1.1}$$

where E is energy and p_x, p_y, p_z are the classical components of momentum. In Eq. 1.1, natural units, where the speed of light is $c = 1$, have been used.

Invariant mass

The invariant mass of a particle, or a system of particles is defined as

$$m_0^2 = p^\mu p_\mu = E^2 - |\vec{p}|^2 \ , \tag{1.2}$$

where p^μ is the sum of the 4-momenta of all particles in the system, $p^\mu = \sum_{i=1}^{n} p_i^\mu$. Hence, for a system consisting of two sets of particles, one with 4-momentum p^μ and one with 4-momentum k^μ the invariant mass is

$$m_0^2 = (p^\mu + k^\mu)(p_\mu + k_\mu) \ . \tag{1.3}$$

Interaction cross-sections and resonances

In particle physics, the interaction *cross-section* of a process is the quantum mechanical probability of that process to happen when two particles collide [5]. It is in general not related to the geometric cross section of the colliding particles. The term cross-section comes from the fact that the interaction probability happens to be expressed in the same unit as a geometric cross-section, namely that of an area. Interaction cross-sections are Lorentz invariant which means that they are the same in all inertial reference frames.

An important property of cross-sections is that they can be dependent on the energy of the colliding particles. That means that two particles colliding with a relative speed of $0.5c$, in general, has a different interaction probability compared to the case where the same two particles collide at $0.9c$. As an example, consider Fig. 1.2 which presents the cross-section for the production of hadrons when electrons and positrons collide as a function of collision energy. There is a large, *resonant* peak at around 80 GeV caused by the Z boson, which acts as an intermediary state in the hadron production, $e^+e^- \to Z \to$ hadrons. The Z boson has never been measured directly, its lifetime is much too short at roughly 10^{-25} seconds, but is inferred to exist through indirect detection of an abundance of hadron production at a collision energy corresponding to its mass. Particles that exist only as intermediary states, like the Z boson is sometimes referred to as resonant particles.

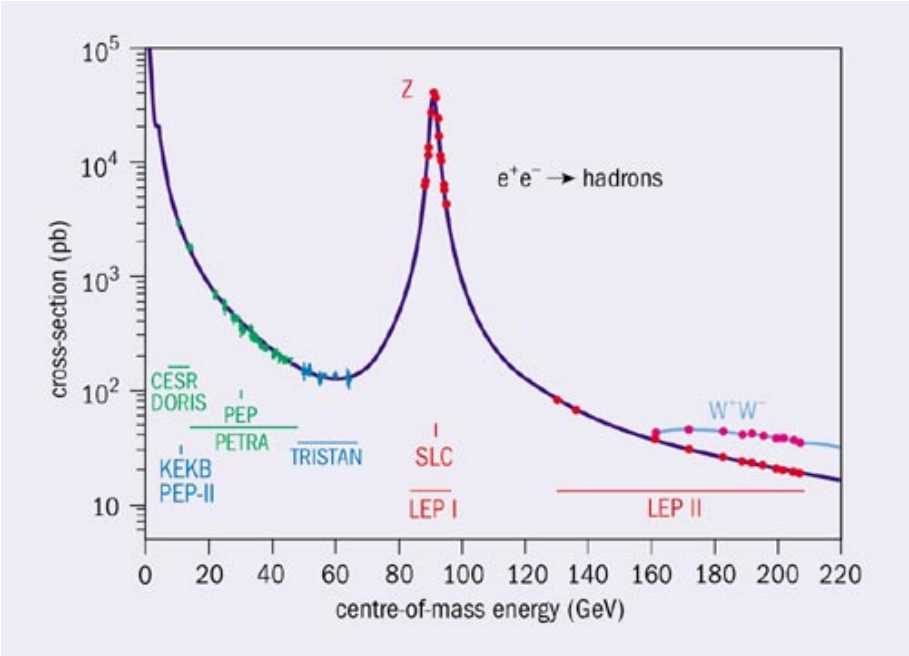

Figure 1.2: The $e^+e^- \to$ hadrons annihilation cross-section, from initial low energy colliders to the maximum energy of the LEP collider at CERN [6].

1.1.3 Collision Events and Hadron Jet Production

A *collision event*, or here sometimes simply referred to as an event, is the information left in the detector from the collision of two protons in the LHC. Since protons are not fundamental particles but made up of one down and two up quarks, it is actually quarks that collide. In a collision, the quarks in a proton can be forced apart by the forces involved and scatter in different directions.

Before moving on to hadron jet production, a phenomenon called *color confinement*, which has little to do with actual colors, has to be introduced. In the field of physics called Quantum Chromodynamics (QCD), quarks and gluons have a property called *color charge* which can have 6 different values: red, antired, blue, antiblue, green and antigreen. Color confinement requires that the color charge of free particles must be zero. That can be accomplished either by combining three particles with {red, green, blue} or {antired, antigreen, antiblue} or by combining any color with its corresponding anticolor. Interactions between quarks are governed by the strong force, which is described in QCD. Now, in particle physics, if there is a force there is a force carrying particle that mediates it. Electromagnetism is mediated by photons, the weak force by W and Z bosons and the strong force by gluons.

Since quarks have color charge they cannot exist freely on their own, they are kept together by gluons. A simple analogy that is often used to describe how gluons keep quarks together is a classical spring where the force of the spring is linearly proportional to its elongation. When two quarks are forced apart, potential energy is built up in the spring and when there is enough energy the spring snaps, converting the potential energy to a quark-antiquark pair. In other words, when quarks are separated far enough it becomes energetically favourable to create a quark-antiquark pair. If the collision event is very energetic this process of quark-antiquark creation can happen many times and gives rise to hadron jets. Hadrons are particles made of 2 or more quarks. The process of hadron jet production is described schematically in Fig. 1.3.

Some hypothesized particles not described by the SM, i.e. Dark Matter (DM) particles, are predicted to decay into two jets, or *dijets*. DM is a form of matter which has not yet been directly detected but its existence is inferred from much experimental evidence. For instance, its gravitational effect has been observed in the so-called Bullet Cluster [7], a system of two colliding galaxy clusters. DM is thought to make up about 85% of the matter in the Universe while the remaining 25% is normal, or baryonic, matter [8, 9]. For further details about the history of DM research a comprehensive review can be found in [10].

Searches for such particles often rely on the appearance of a resonant peak (similar to the resonant peak of the Z boson in Fig. 1.2) on top of a smooth background. In the case that the final product is two jets and there exists a resonance, a resonant peak can be seen if the cross-section for dijet production is plotted as a function of the dijet mass, that is the invariant mass of the two jets. Fig. 1.4 shows an example of a process involving DM which would result in a dijet resonance.

In practice, the dijet cross-section is not measured directly. Instead the number of dijet occurrences in different dijet mass intervals are counted in order to produce a histogram. If there exists a dijet resonance in any of those mass intervals, an abundance of dijets will have been produced at that mass level and a peak can be seen in the histogram.

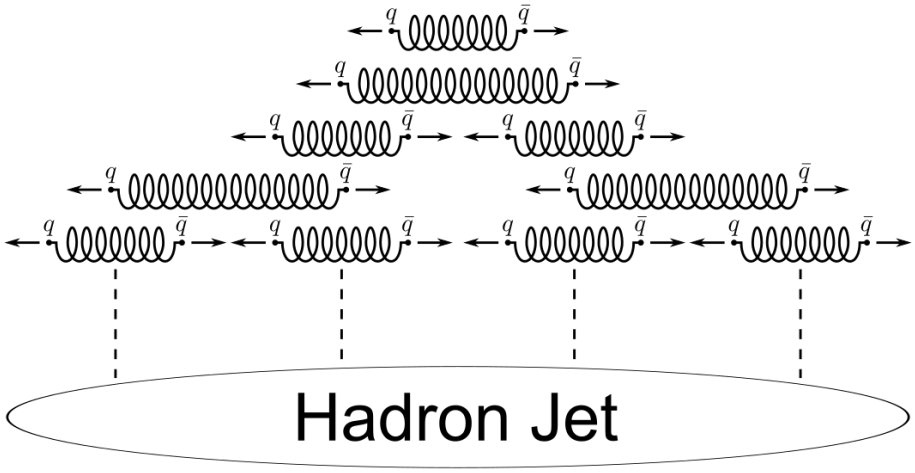

Figure 1.3: Sketch describing the creation of a hadron jet. [11]

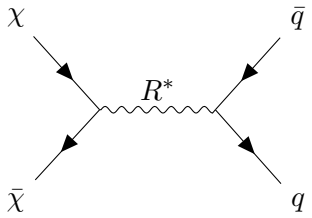

Figure 1.4: Example of a process involving dark matter (χ, $\bar{\chi}$) and a dark matter mediator (R^*).

1.1.4 The ATLAS experiment and its trigger system

ATLAS [12] stands for A Toroidal LHC ApparatuS and is the largest experiment at CERN. It is a general purpose particle detector designed to detect the particles that are created in the proton-proton collisions produced by the LHC.

There are approximately 1.7 billion collisions occurring inside the ATLAS detector, each second [13]. Most of these collisions are not interesting and therefore will not be sent to storage. In addition, current technological limitations make it impossible to process and store the enormous amount of data that gets produced, even in the cases where that data is interesting. Hence, ATLAS uses a so called *trigger system*, consisting of two levels, one hardware level and one software level. The hardware trigger saves at most 10^5 collision events per second. Thereafter, the software trigger performs an additional analysis and picks out about 10^3 events per second which are sent to a data storage system for later analysis [13].

This data reduction impairs the potential of discovering new particles that are buried in high-rate backgrounds. Therefore, a new technique called "Trigger Level Analysis" (TLA) has been brought forward by scientists in the ATLAS Collaboration (with the Lund and OSU groups among others), following the example of the CMS and LHCb Collaborations [14, 15], to only record high-level information necessary to perform searches for those new particles. Since the size of each event is smaller, more events can be recorded, as shown in Fig. 1.5.

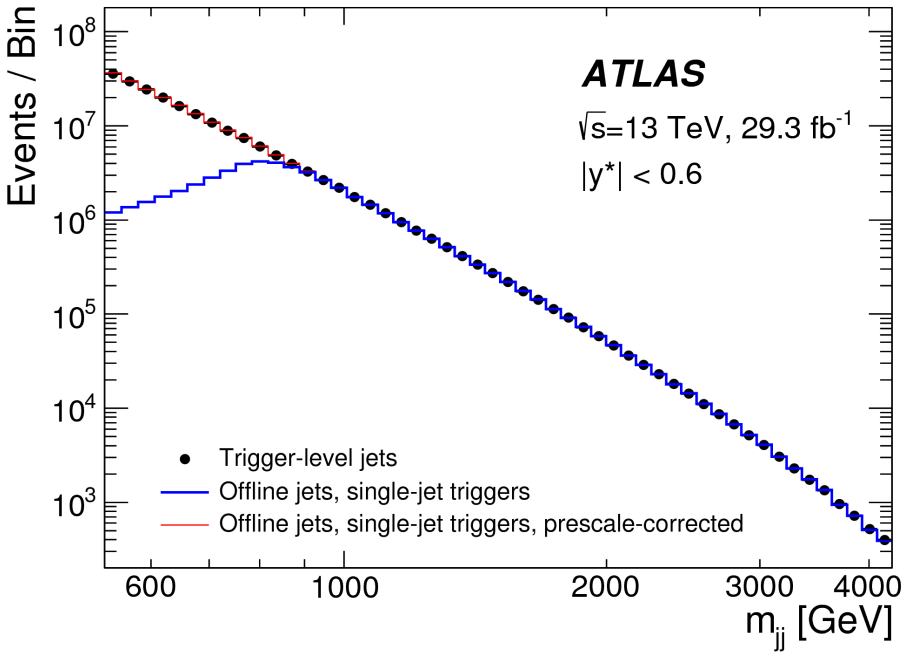

Figure 1.5: Improvement in number of events recorded with the TLA technique (black) with respect to the standard data gathering techniques (blue line) [16].

A further reduction of the current event size will allow for searches that were not previously possible, for example for even lower mass dark matter mediators [17].

The focus of this thesis is to understand whether it is possible to further reduce the size of previously mentioned collision events using machine learning (ML) techniques for data compression. This has never been tried before within the ATLAS experiment and it is an interesting forward-looking study for future experiments.

It will also be interesting to treat this study as a proof-of-principle for future data compression techniques for the ATLAS experiment. In light of the planned accelerator and detector upgrades in 2026, the techniques used in this report may help solving the problem of needing much more storage space than in the past due to the increase of the size of the dataset, as shown in Fig 1.6.

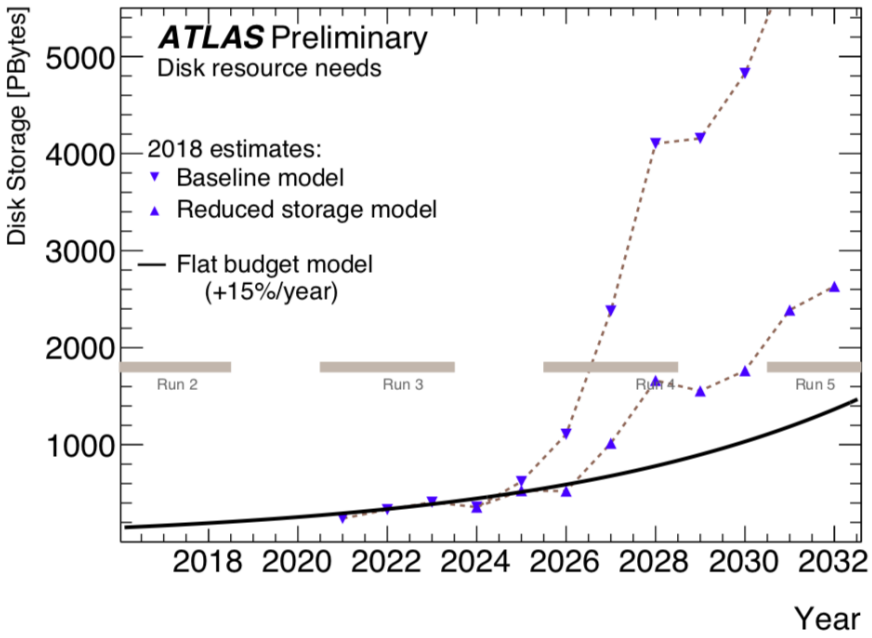

Figure 1.6: Estimated future disk space required for storage of experimental data captured by ATLAS. Run 2, 3, 4 and 5 are planned operational periods of the LHC. The runs are interleaved by periods where the accelerator is taken out of operation for technical maintenance and upgrades. Plot from [18].

1.2 Artificial Neural Networks and Deep Learning

1.2.1 The basics

Artificial Neural Networks (NNs) are widely used in modern statistical prediction and machine learning and have risen to state of the art performance in areas such as image classification and natural language processing (NLP). They have also been used successfully in science, for example aiding drug development by predicting the behavior of medical compounds [19], helping doctors to diagnose patients [20] and physicists to search for exotic sub-atomic particles [21]. They are loosely based on biological neural networks of which they can be seen as a simplified model. Like a brain, they are made up by a number of neurons, or computational nodes, that pass information between each other through a network of connections. The simplest possible NN consists of a single neuron as in Fig. 1.7 and is sometimes referred to as a perceptron. The perceptron was first introduced as a probabilistic model of computation and memory storage in the brain [22] and has been used in many different variations since then. The perceptron pictured in Fig. 1.7 takes an n-dimensional vector as input, computes a weighted sum according to the weights, $w_1, w_2 \dots w_n$, and feeds the resulting sum through a nonlinear activation function $f(x)$ which gives the final scalar output y. This is described in Eq. 1.4.

$$y = f(\sum_{i}^{n} w_i x_i) \tag{1.4}$$

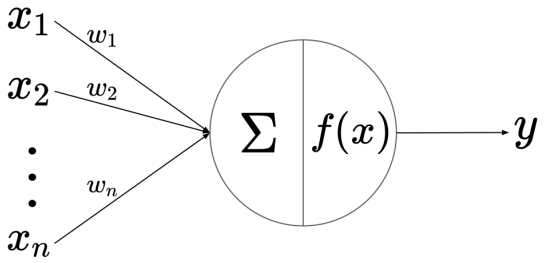

Figure 1.7: Illustration of a single neuron accepting an n-dimensional vector as input and giving a one-dimensional output y.

Training neural networks correspond to adjusting the weights in order to minimize a given loss function and will be discussed further in Sec. 1.2.2.

A single computational node like the one in Fig. 1.7 is, not surprisingly, incapable of solving complex machine learning problems. The power of NNs lie in connecting many of these nodes into large networks where they can interact with each other and are able to implement complex nonlinear functions. Fig. 1.8 shows such a network, consisting of one input layer of size 6, two *hidden* layers of size 10 and finally a two-dimensional output layer. A NN is commonly considered deep if it has one or more hidden layers and deep learning (DL) is thus the practice of using deep neural networks (DNNs) for machine learning.

How the performance of NNs are measured depend on the problem they are meant to solve. For classification problems, a natural performance measure is the classification accuracy, i.e. the fraction of data points that are correctly classified by the NN. For regression problems, a common performance metric is the mean squared error (MSE) of a series of data points, defined as

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{P}(d_{ni} - y_{ni})^2 \ , \tag{1.5}$$

where y_{ni} is the ith output of the NN for the nth data point, d_{ni} is the correct value of that output, P is the size of the input space and N is the number of data points in the dataset.

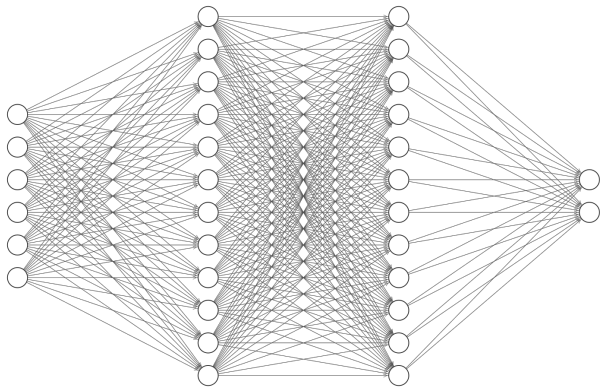

Figure 1.8: Example of a NN with two hidden layers taking six variables as input and giving two output variables.

The choice of which activation function to use can have a big impact on the performance

of NNs. Some common alternatives are presented in Tab. 1.1 and Fig. 1.9. The logistic function and the hyperbolic tangent both takes input from $(-\infty, \infty)$ and output values in a finite range, which, depending on the problem at hand, can be a useful property. For instance, in the case of binary classification, the output of the NN can be interpreted as the probability that a specific input belongs to one of the two classes, hence the logistic function, which produces output in $(0, 1)$ would be a good choice for the output activation. The logistic and hyperbolic tangent activations often work well for shallow NNs but if used in deep networks they can suffer from the so-called vanishing gradient problem, described in [23], which arises, in part, as a consequence of the very small gradients of these functions for inputs larger than ~ 10. During backpropagation [24] the gradients can get smaller and smaller the further back through the network they propagate which results in neurons in earlier layers learning slower than neurons in later layers. This, in turn, leads to slower overall training and worse performance. Another drawback of these activation functions is the computational cost associated with computation of the exponential function.

The Rectified Linear Unit, or ReLU, attempts to solve the issues facing the logistic and hyperbolic tangent activations. It is computationally cheaper to use since all that is needed is to take the maximum of the zero and the input. Furthermore, since it is the identity function for inputs $x \geq 0$ it doesn't suffer from the vanishing gradient problem. However, mapping all negative values to identically zero can result in undesirable consequences. The so-called dying ReLU problem [25] arises when the input to a ReLU activation gets stuck at a negative value causing the ReLU to always output zero. If this happens the ReLU is unlikely to recover from this state since it blocks the flow of gradients during backpropagation. Since gradients are multiplied together during backpropagation (according to the chain rule of derivatives), they can be said to flow back through the network. Now, if one of the gradients is always zero due to a dead ReLU activation, no weights before that dead ReLU will be updated. In spite of this, ReLU is one of the most popular activation functions in DL and often produce great results.

In an attempt to solve the dying ReLU problem the Leaky-ReLU was introduced. For positive values it is identical to the ReLU but for negative values it doesn't output zero but rather a small negative value. This allows gradients to always flow back through the entire network during backpropagation.

Table 1.1: Commonly used activation functions.

Name	Formula
Logistic function	$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent	$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
ReLU	$f(x) = \max(0, x)$
Leaky ReLU	$f(x) = \max(0.1x, x)$

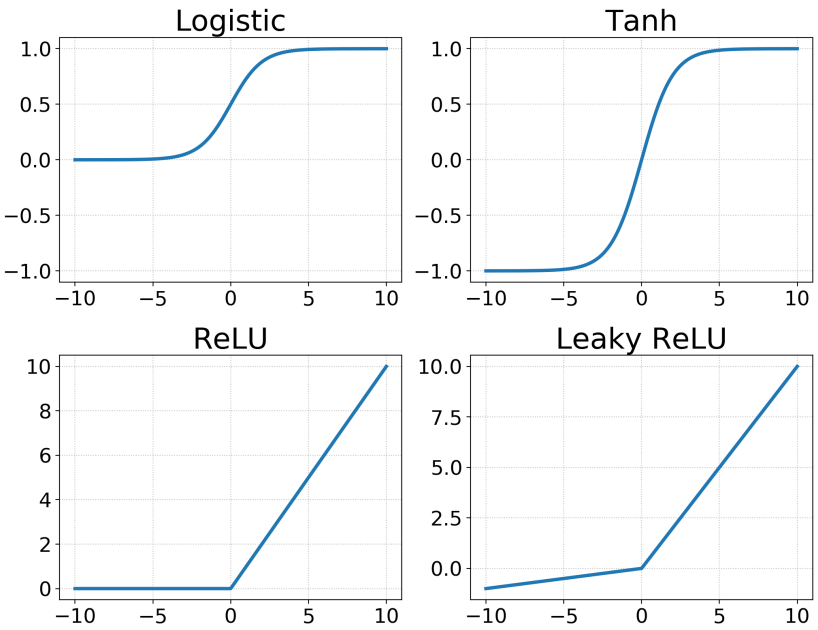

Figure 1.9: Some common activation functions.

1.2.2 Training Neural Networks

The data

Data is one of the most important components when training machine learning models and in general the more that is available the better. Furthermore, it is desirable to train models that generalize well, i.e. models that perform well on previously unseen data. In order to be able to evaluate the generalizing power of a trained model it is therefore necessary to put a portion of the available data aside before training as in Fig. 1.10. This portion of data is usually referred to as the validation data or the validation set, in contrast to the training data or training set which is used during training. What fraction of the available data that is to be used to create a validation set depends on the problem at hand. It is however a good idea to make the validation set at least large enough to compute an evaluation metric to desired statistical significance. The consequence of making the validation set too small is that the value of the chosen evaluation metric computed from the validation set cannot be trusted. Therefore a compromise often has to be made between the benefit of training on more data and the crucial ability to evaluate the performance of the trained model. [26]

Figure 1.10: The available data is split into training and validation sets.

Defining a learning objective – The loss function

The loss function is most commonly a mathematically defined function,

$$f : \mathbb{R}^P \longrightarrow \mathbb{R} \tag{1.6}$$

that takes the P-dimensional output of a NN as its input and compares it with some "correct" answer to produce a real number representing how far away the NN's output is to the correct answer. Thereafter an optimization algorithm is run in order to minimize the value of the loss function. The loss function can be said to define an *optimization landscape*, a high-dimensional surface on which the optimization algorithm's task is to find the lowest point, i.e. the point where the value of the loss function is minimized. For larger NNs with many trainable parameters this can be a very difficult problem to solve and it is often not possible to determine whether a global or local minimum has been reached at the end of training.

Optimization algorithms

There exists a wide range of different optimizers for use in deep learning, all with different strong and weak points. They all aim to minimize a given loss function and are more or less advanced variants of an algorithm called gradient descent. In short, gradient descent is an algorithm in which the gradient of the loss function with respect to the weights of the NN is computed, whereafter all weights in the NN are adjusted slightly in the direction opposite to the gradient as described in Eq. 1.7 where $\vec{w}$ is a vector containing all the model weights, α is the learning rate, $f(\cdot)$ is the loss function, ∇_w is the gradient operator with respect to all trainable weights and m is the number of data points or training examples.

$$\vec{w}_{t+1} = \vec{w} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_w f(\vec{x}_i, \vec{w}_t) \tag{1.7}$$

The learning rate, α, is an example of a hyperparameter, which can be defined as any parameter of the model that is not learned during training. There is no formal proof that gradient descent will find a good minimum of the loss function but if hyperparameters are chosen correctly, it often does in practice. The choice of learning rate will be discussed further below.

In the gradient descent algorithm, all available data is used to compute the average gradient used in each optimization step. If the dataset is large this can be computationally expensive and as a result the modified version called stochastic gradient descent (SGD) [27] is often used. In SGD, only a subset of the available data is used to compute the average gradient before each optimization step. Such a subset is referred to as a mini-batch and the size of the mini-batch is called the *batch size* and can be considered as a hyperparameter when training NNs. When the entire training set has been used once, the network has been trained for one *epoch*. Hence, with gradient descent there is one optimization step per epoch while there are many optimization steps per epoch in SGD.

Even SGD can sometimes be slow to converge, especially if the gradients of the loss function are small. A method designed to alleviate this issue is called *momentum* [28] and works by keeping a decaying moving average of past gradients to keep the gradient descent moving in their direction. This is similar to an object having inertia and the concept of momentum in physics which is where this method got its name from.

Using SGD with momentum changes the update rule of Eq. 1.7 to

$$\vec{v}_t = \epsilon\vec{v} - \alpha\frac{1}{m}\sum_{i=1}^{m}\nabla_w f(\vec{x}_i, \vec{w}_t) \ , \tag{1.8}$$

$$\vec{w}_{t+1} = \vec{w}_t + \epsilon\vec{v}_t \tag{1.9}$$

where $\epsilon \in [0, 1)$ is a hyperparameter called the momentum parameter which controls how quickly the contribution of previous gradients to the moving average decays and α is, like before, the learning rate.

The Adam optimizer, first introduced in 2014 [29] has been used throughout the course of this project and combines the use of momentum with individual learning rates. It is based on the idea that not all weights in the model are wrong by the same amount and should therefore not be updated using the same learning rate. There is still a global learning rate, defined by the user, but the Adam algorithm uses this to compute individual learning rates for each trainable weight in the NN. For more details on the workings of Adam, the reader is referred to [29].

Choosing the learning rate

The learning rate is sometimes referred to as the most important hyperparameter to get right in DL. In spite of this, finding a good learning rate is often done by trial and error and can be time consuming. A cyclical learning rate policy that has proven to work well is the *1cycle* policy [30]. In its simplest form, the 1cycle policy only requires the user to choose a maximum and minimum learning rate. Good minimum and maximum values can be chosen using the learning rate range test described in [31]. The 1cycle policy will be discussed further in Sec. 2.3 where its use in the context of this project is described.

1.2.3 Model capacity, underfitting and overfitting

The capacity of a model is its ability to represent or fit a wide range of functions. In the context of deep learning, more nodes and more layers imply higher model capacity. Choosing a model with appropriate capacity for the task at hand can have a big impact on the performance of said model. A model low on capacity may struggle to solve the problem and result in underfitting while a model with too high capacity may be prone to overfitting.

Fig. 1.11 illustrates the behaviour of the training and validation losses as a function of model complexity. The same behaviour can be seen as a function of the number of training epochs. Typically the training loss continues to decrease when training for longer while at some point the validation loss starts increasing. At this point the generalizing ability of the network is getting worse and instead of learning general patterns from the data the network is memorizing the specific examples in the training set. A simple technique to prevent overfitting is called *early stopping* which simply ends the training when a given stopping criterion, for instance that the validation loss has not decreased for a given number of epochs, is fulfilled.

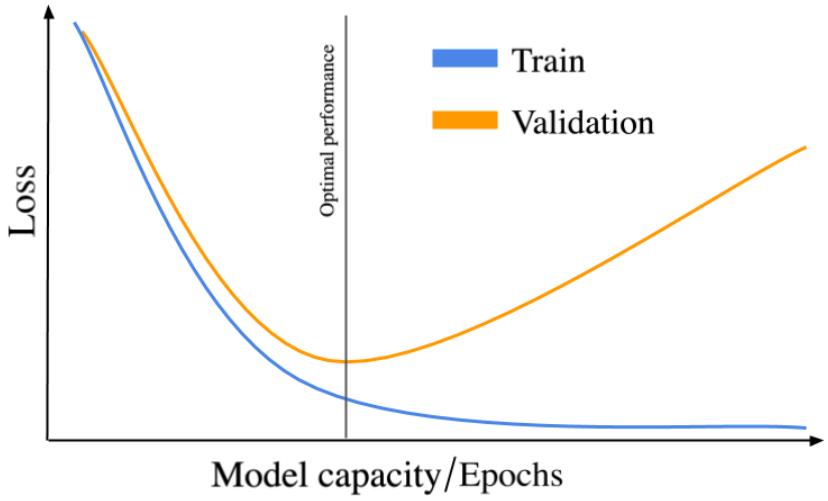

Figure 1.11: Sketch illustrating the behaviour of the training and validation losses as a function of model complexity or number of training epochs.

1.2.4 Regularization

Regularization is a method used to prevent overfitting and is crucial to successfully training DNNs. The previously discussed technique of early stopping can be seen as a simple regularization method since it is designed to stop training the network when it starts to overfit. There exist many other regularization techniques, some of which will be described in this section.

Dropout

One of the simplest regularization techniques used in DL is based on randomly turning off some proportion, p, of neurons in one or more layers during training. This technique is called dropout [32, 33] and works by encouraging the extraction of features that are useful on their own, without relying on other extracted features. In other words, dropout prevents co-adaption of feature detectors [33]. It can be seen as a way to limit model capacity since some neurons are turned off during training and can't be used, thereby limiting model capacity. When training is complete, however, all neurons are used for inference. Dropout will not be used in this thesis project.

Weight decay

Weight decay is based on the idea that NNs with large weights can be very sensitive to their input and might not generalize well. Large weights can occur when training for a large number of epochs since then the weights are updated many times and have the opportunity to keep growing. Weight decay, Eq. 1.10, is sometimes confused with L_2-regularization, Eq. 1.11, due to the fact that they become equivalent when using SGD with a certain set of hyperparameters. However, if adaptive gradient algorithms, e.g. Adam, are used they are not equivalent and it has been shown by [34] that in the case of Adam, proper weight decay outperforms L_2-regularization. Therefore it is important to separate the two. In

weight decay, the weights are made to decay exponentially according to

$$w_{t+1} = (1 - \lambda)w_t - \alpha \nabla f(w_t) \tag{1.10}$$

where λ controls the rate of decay per optimization step, α the learning rate and $\nabla f(w_t)$ the gradient of the loss function with respect to the weights as computed on the t-th step.

L^2 regularization, on the other hand, simply penalizes large weights by adding the sum of squares of all weights multiplied by a factor, β, to the loss function as described in Eq. 1.11. As mentioned previously, when using SGD, weight decay and L_2-regularization are equivalent if the correct hyperparameters are used. Specifically, [34] shows that choosing $\beta = \alpha/2\lambda$, weight decay and L^2-regularization are equivalent.

$$f_{L^2}(w) = f(w) + \beta \sum_i w_i^2 \tag{1.11}$$

Some popular machine learning frameworks implement weight decay as L_2-regularization wherefore users should always make sure they are using the regularization method they intend to and not trust that the chosen framework implements weight decay as they expect. For instance, in PyTorch, the optimizer `torch.optim.Adam` has a weight decay parameter, `wd`, which is actually L_2-regularization. In order to use proper weight decay the AdamW optimizer, `torch.optim.AdamW`, introduced in PyTorch after the publication of [34], should be used.

Batch normalization

Batch normalization was first introduced in 2015 [35] and has since become standard practice to use in DNNs. It helps both in terms of stability during training, allowing for higher learning rates and therefore faster training, and in terms of regularization [36]. When batch normalization was first discovered it was thought to work by reducing the internal covariate shift which the authors of the original batch normalization paper [35] define as the change in the distribution of activations due to the change in network weights during training. To explain further, since the output of one layer is given as input to the next layer, updating the weights in the first layer results in changing the distribution of input variables for all subsequent layers.

However, in 2018, [37] showed that the actual reason why batch normalization helps training a DNN is that it makes the optimization landscape smoother and thereby the gradients more predictable, allowing for higher learning rates and faster training.

The algorithm for batch normalization as described in [35] is provided in Algorithm 1 where ϵ is a small constant added for numerical stability.

Require: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Ensure: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch. [35]

1.2.5 Autoencoders

An autoencoder (AE) is a neural network which tries to implement an approximation to the identity function. It generally consists of an encoder, a so called latent space, which is smaller in size than the input layer, and a decoder. The encoder encodes the information present in the input into the latent space and the decoder reconstructs the original input as best it can. If the AE is able to reconstruct its input well, most of the information necessary to reproduce the input is contained in the latent space. This means that the latent space representation can be used as a compressed representation of the input and could be stored along with the decoder network to reconstruct the data.

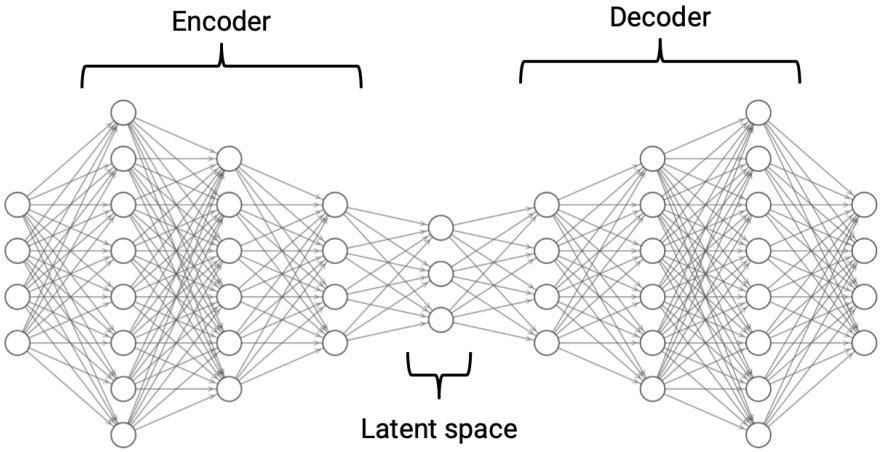

Figure 1.12: Example sketch of an AE with a 3-dimensional latent space.

Moreover, an AE can be used to detect anomalous data in a series and works as follows. The network is first trained on data which is known not to be anomalous. If then the network is presented with a new data point that differs in some significant way from the training data, it will not be able to provide a faithful reconstruction at the output layer and hence the data point is considered anomalous (see [38] for an example in fraud detection). In other words, if the *reconstruction error* of a data point is large it is

considered anomalous.

It has been shown that AEs can be used to reduce the dimensionality of data and to extract useful feature representations [39, 40]. In fact AEs are doing transformations similar to principal component analysis (PCA) even though there are some important differences [41]. For instance, while PCA is restricted to linear transformations an AE can perform more complex non-linear transformations. Moreover, the results of PCA are easier to interpret, in part due to the fact that the principal components are orthogonal and ordered according to the amount of variance they explain.

A commonly used loss function for training AEs is the MSE, already defined in Eq. 1.5. In the case of AEs, the correct output is always equal to the input so the MSE can be written as

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{P}(y_i(\vec{x}_n) - x_{ni})^2 \ , \tag{1.12}$$

where y_i is the ith output element, x_{ni} is the ith input element in the nth input vector, $\vec{x}_n$, P is the length of $\vec{x}_n$ and N is the number of data points in the dataset. Minimizing the MSE is then equivalent to minimizing the difference between the input and the output.

The focus of this work is the use of AEs for compression by using the latent space as a compressed representation of the data. The ratio of input variables to latent space dimensions determines the compression ratio and is expected to highly affect the reconstruction accuracy. Some important aspects to consider when evaluating AEs for data compression in physics are training time, encoding/decoding time, model storage size and the evaluation metrics presented in Sec. 1.2.6.

1.2.6 Evaluation metrics

In this section the evaluation metrics used to measure the performance of AEs in this project is presented. An obvious metric to consider is the validation loss, e.g. the MSE, but that does not give much qualitative information about how well the jets are reconstructed. Therefore other metrics such as the difference between input and output,

$$x_{out} - x_{in} \ , \tag{1.13}$$

the relative difference of the same,

$$\frac{x_{out} - x_{in}}{x_{in}} \ , \tag{1.14}$$

and the mean and standard deviations of them both were used.

Chapter 2

Method , Results and Discussion

This project aims to shed light on how useful AEs can be for compression of experimental high energy physics data and to evaluate the performance of explored NN architectures and hyperparameter combinations. This is the first time AEs are used for data compression within the ATLAS collaboration, but there are examples in the LHCb collaboration [42].

The project can be divided into two stages. In the first stage an AE was used to compress the 4-momentum of hadron jets from experimental data into 2 or 3 latent variables while in the second stage 27 variables from a derived AOD with TLA data were reduced to a varying number of latent space variables. The decision to start experimenting with a small network was motivated by the fact that it is easier to work with fewer variables rather than many, and that it is better to try a new method on complex physics objects such as jets starting from their simplest characteristics. A small network requires less data preprocessing and since a smaller number of input variables leads to a smaller overall model, there is less need for computational power during training. After drawing wisdom from this first attempt, the input was extended to include 23 additional variables (27 in total) from TLA AODs. AOD, which stands for Analysis Object Data, is a data format used by the ATLAS Collaboration for analysis and contains variables which describe the characteristics of jets. The methods and tools used to design, train and evaluate the AEs as well as their performance are presented in the following sections.

2.1 The Data

The datasets in this project originate from three different files as presented in Tab. A.1 in Appendix A.1.

To begin with, only the variables from the 4-momentum (transverse momentum p_T, pseudorapidity η, azimuthal angle ϕ, energy E) of all leading jets were extracted from a sample of experimental data provided by Bryan Reynolds at the Ohio State University (OSU). By leading jet is meant the jet with highest transverse momentum within one collision event. The reason for extracting only the leading jets is no other than reducing the size of the dataset so as to make fast experimentation easier in the start of the project. This first dataset will be referred to as $\mathcal{D}_A$. Later, when the networks used were expanded to take 27 input variables, all of which are described in Appendix A.2, another set of experimental data was used. Furthermore, all jets, not only the leading ones, were included in order to make the dataset more realistic. This dataset will be referred to as $\mathcal{D}_B$. Limitations in computer memory made it cumbersome to handle $\mathcal{D}_B$ which made the compute nodes used for training crash due to insufficient memory. This could have been

overcome by only loading part of the training data into memory at any one time but for the sake of simplicity another dataset, $\mathcal{D}_C$ was created by randomly selecting 10% of the jets present in $\mathcal{D}_B$. Note that for getting the best possible end result it is probably better to use all available data. However, at the time of creating $\mathcal{D}_C$ no sign of overfitting had been seen, so insufficient training data was not considered a major problem. Furthermore, the increase in the number of input variables from 4 to 27 significantly increased the computational cost, and therefore also the time needed for training, making a smaller dataset more attractive for fast experimentation and hyperparameter optimization.

It is common to perform a jet cleaning procedure before analysis in order to avoid using jets coming from noise, cosmic showers or beam induced background. Also in this case, a cleaning procedure was used. The cuts described in [43], which was derived with over 99% efficiency of selecting good jets for jets with $p_T > 20$ GeV, were used as a starting point. Some of the cuts were made looser by a factor of 5 and additional cuts on variables[1] not mentioned in [43] were also made. In total 4091 out of 956154 jets were removed (about 0.4%) by the cuts which are described in Tab. 2.1. These cuts were made in order to filter out fake jets that could have negatively affected the training without actually being useful for the data analysis. A potential drawback of removing jets is that the AEs wont have seen fake jets during training and therefore may not be able to reconstruct them with good accuracy. This, however, is not a major drawback since measures are always taken to not use fake jets for physics analysis.

Last but not least a signal Monte-Carlo (MC) sample (from simulation) of a DM mediator decaying into dijets was used for further testing of the AEs abilities. This dataset will be referred to as $\mathcal{D}_{MC}$ and it was used for an independent test of the AEs reconstruction accuracy when they only had been trained on the experimental background data in $\mathcal{D}_C$. Tab. 2.2 summarizes the different datasets.

Table 2.1: Jet cleaning criteria.

#	Cleaning criteria
1	$\lvert$`HECQuality`$\rvert > 0.5$ and $\lvert$`HECFrac`$\rvert$ and $\lvert$`AverageLArQF`$\rvert > 0.8$
2	$\lvert$`NegativeE`$\rvert > 300$ GeV
3	`AverageLArQF` > 0.8 and `EMFrac` > 0.95 and `LArQuality` for jets with $\lvert$`eta`$\rvert < 2.8$
4	`EMFrac` < 0.05 and $\lvert$`eta`$\rvert < 2$
5	`Timing` > 125 ns
6	`LArQuality` > 4
7	`HECQuality` > 2.5
8	`Width` > 5 or `Width` $= -1$
9	`WidthPhi` > 5
10	`OotFracClusters5` < -0.1
11	`OotFracClusters10` < -0.1

[1] `Width`, `WidthPhi`, `OotFracClsters5` and `OotFracClsters10`.

Table 2.2: Dataset names and notes.

Name	Number of jets	Description
$\mathcal{D}_A$	1937900	Experimental data courtesy of B. Reynolds's
$\mathcal{D}_B$	11951922	Derived AOD with TLA data
$\mathcal{D}_C$	1195192	10% of $\mathcal{D}_B$
$\mathcal{D}_{MC}$	10000	Monte-Carlo simulated signal sample

2.2 Autoencoders with 4 variables as input

The architecture design of NNs typically involves a proportion of trial and error. Some clues about which architectures are worth trying and might work well on the specific problem at hand can be found by studying the results of published research. Ultimately, however, one has to give up on finding the optimal architecture and settle for finding one that performs well enough. Even if the optimal architecture would be found, there is often no way to prove that it is optimal.

In the first few attempts at training AEs the architecture was kept simple and small. Layers of up to 8 nodes were used in networks with depths from 1 to 9 hidden layers. Increasing the number of nodes per layer immediately resulted in a significant improvement of the reconstruction accuracy, so the narrower networks were not investigated in further detail. As expected, increasing the width (see Fig. 2.1) of the networks affected the required training time negatively making time the constraining factor of how wide it was practical to make the AEs. After initial experimentation, limited hyperparameter grid searches were performed for a number of different architectures with widths varying from 50 to 200 neurons. Fig. 2.1 illustrates a general four variable input AE with a three dimensional latent space.

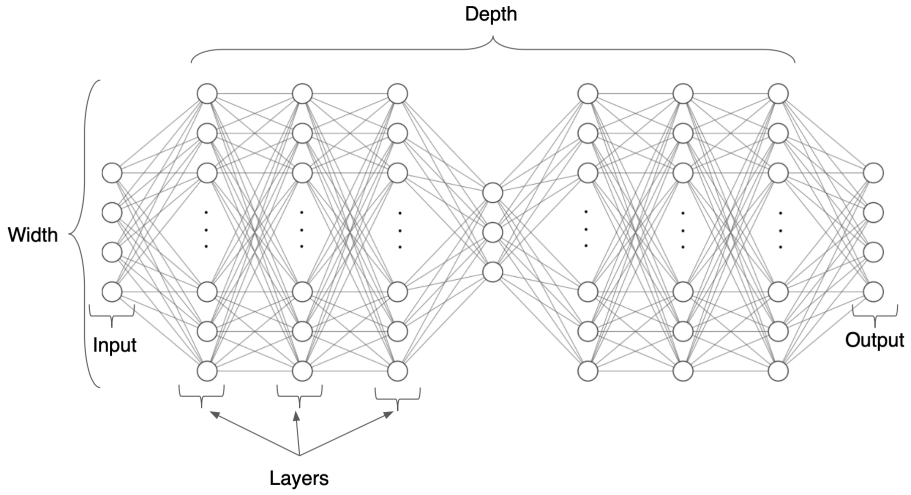

Figure 2.1: Sketch of a general 4 variable input AE with a 3 dimensional latent space. Different widths and depths were explored. There was not necessarily the same number of nodes in each layer, although the architecture was always symmetric around the latent space layer.

The Python machine learning and neural network library PyTorch [44] was used throughout the project. The fastai [45] library, built on top of PyTorch, which provides a range of useful functions was also used. All code used in this project is available on

GitHub [46].

The hyperbolic tangent was chosen as the activation function for all layers except for the output where a linear activation function was used. PyTorch's `AdamW` optimizer was used with varying learning rates. Also the batch size and weight decay parameters were varied. Dropout (Sec. 1.2.4) was tested but immediately resulted in worse performance so was quickly abandoned. The same goes for batch normalization (Sec. 1.2.4).

After some initial trial and error the trained AEs started performing well. The best 4 variable input network that was trained during the project had 7 hidden layers with 200-100-50-3-50-100-200 nodes in each hidden layer and 4 nodes in the input and output layers. The performance of this AE is presented in the results section below. It was trained using a batch size (Sec. 1.2.2) of 256, weight decay parameter of 10^{-6} and a varied learning rate. For the first 10 epochs, a learning rate of 10^{-7} was used, then followed 10 epochs with 10^{-4} and finally 2000 epochs with 10^{-6}. In initial stages where different network architectures were investigated, the number of epochs used were usually much smaller, around 200 or so. However, when a good architecture and hyperparameters had been found, the following strategy was used.

- Start with a small learning rate to get the weights moving in the right direction
- Train at a relatively high learning rate for a few epochs to quickly reduce error
- Finally train for a very long time with a low learning rate

The last step in this strategy is designed to reach as low validation loss as possible without regard to required training time. The idea to start with a low learning rate came from a blog post [47] about the *1cycle* policy [30]. The 1cycle policy implemented in the fastai library was used later in this project for training the 27 variable input AEs and will be discussed in Sec. 2.3.

2.2.1 The data

Heavily asymmetric distributions like the ones of p_T and E seen in Fig. 2.2 can be problematic when normalizing the data. In Fig. 2.3 it can be seen that even after standardization p_T and E are quite skewed with a few extreme values being about an order of magnitude larger than most. This proved not to be critical for the performance of trained AEs. Nevertheless, out of curiosity and the hope of even better performance a customized normalization using the logarithm of p_T and E was investigated [48]. It can be seen in Fig. 2.4 that using this custom normalization leads to all four variables of all jets to be the same order of magnitude. The normalization was done according to

$$\begin{aligned} p_T &\longrightarrow \log_{10}(p_T) \\ E &\longrightarrow \log_{10}(E) \\ \eta &\longrightarrow \eta/3 \\ \phi &\longrightarrow \phi/3 \ . \end{aligned} \tag{2.1}$$

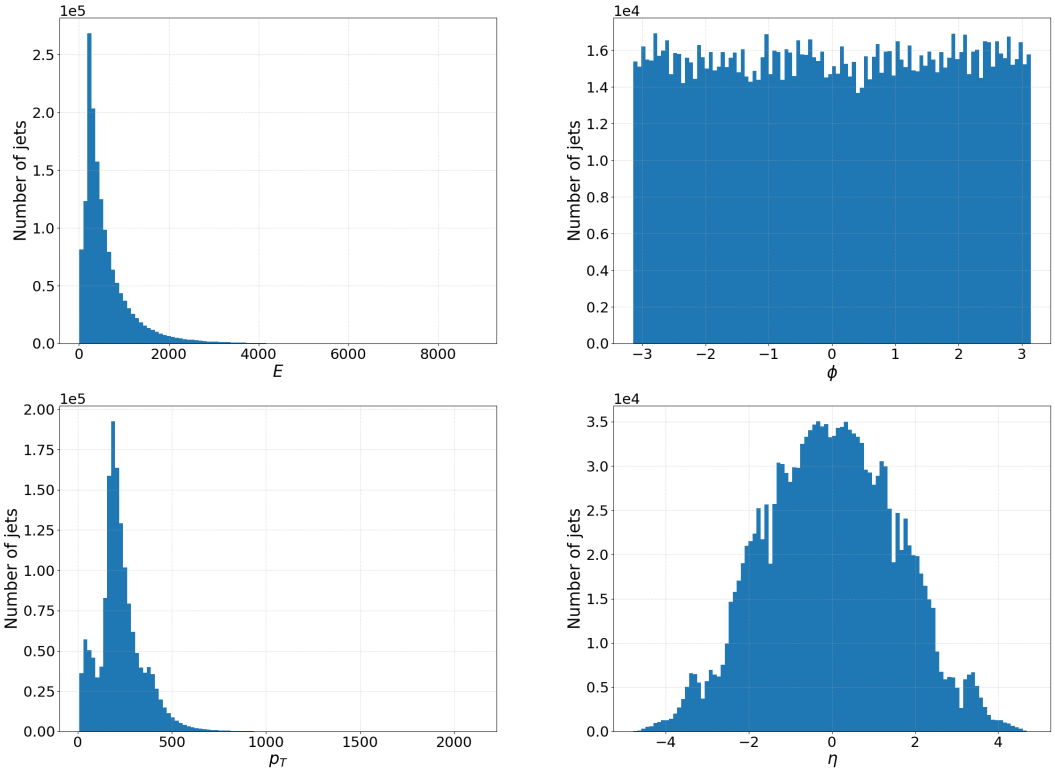

Figure 2.2: Histograms of the four variables in $\mathcal{D}_A$ before standardization.

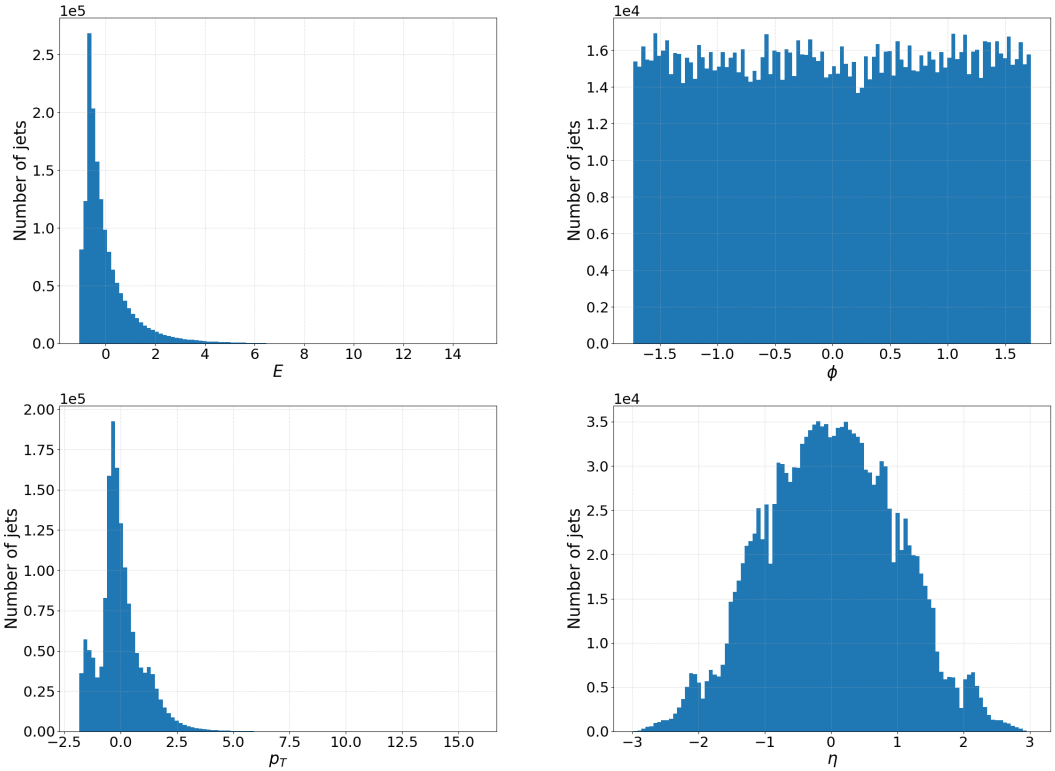

Figure 2.3: Histograms of the four variables in $\mathcal{D}_A$ after standardization. Note the different scale of the E and p_T axes in comparison to Fig. 2.2

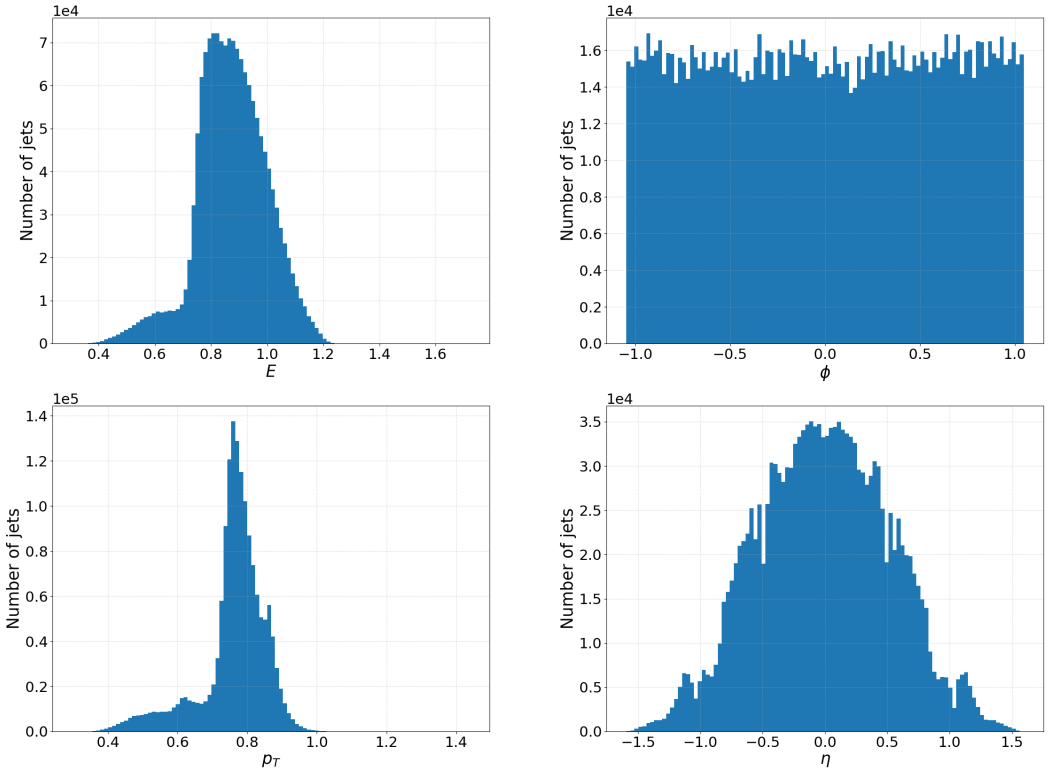

Figure 2.4: Histograms of the four variables in $\mathcal{D}_A$ after performing a custom normalization according to $\{p_T, E\} : x \to \log_{10}(x)$ and $\{\eta, \phi\} : x \to \frac{x}{3}$.

2.2.2 Results and discussion of the $4 \to 3$ AE

Results presented here are from the best performing 4 variable input AE with a 3 dimensional latent space. This gives a compression ratio of $\frac{4}{3} \approx 1.33$ and should be seen as a first proof of concept using a small AE. The MSE validation loss was $5.246327 \cdot 10^{-7}$ on the validation part of $\mathcal{D}_A$. The results presented here are produced by an AE trained on standardized data, not the custom normalization of Eq. 2.1 which proved not to have a significant impact on reconstruction performance. It is possible that normalizing according to Eq. 2.1 could result in slightly better performance than standardization but this was not investigated further since the goal of the 4 variable input AE was not to achieve optimal performance but rather to be a first proof of concept.

The plots in Fig. 2.5 present the distributions of the AE output on top of the original input. The distributions are so similar it is hard to see that there are two different distributions in each plot. The plots in Fig. 2.6 show the relative difference between the output and the input. Here, the x-axis stretches from -2% to 2% and most residuals are much smaller than 1%.

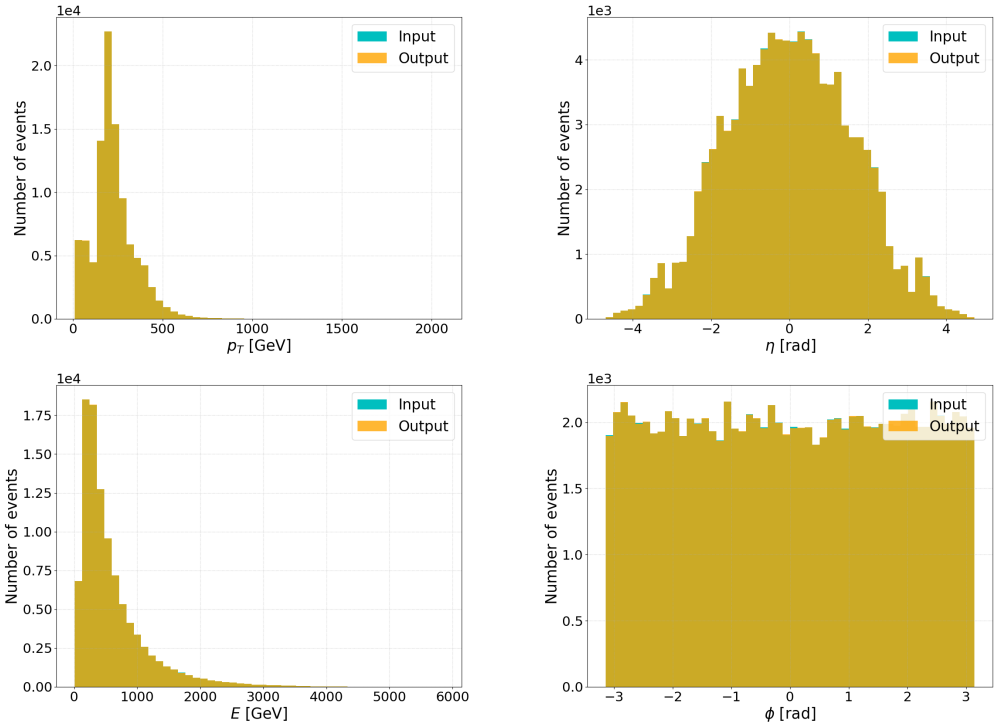

Figure 2.5: Variable distributions of the 3D latent space AE output on top of input.

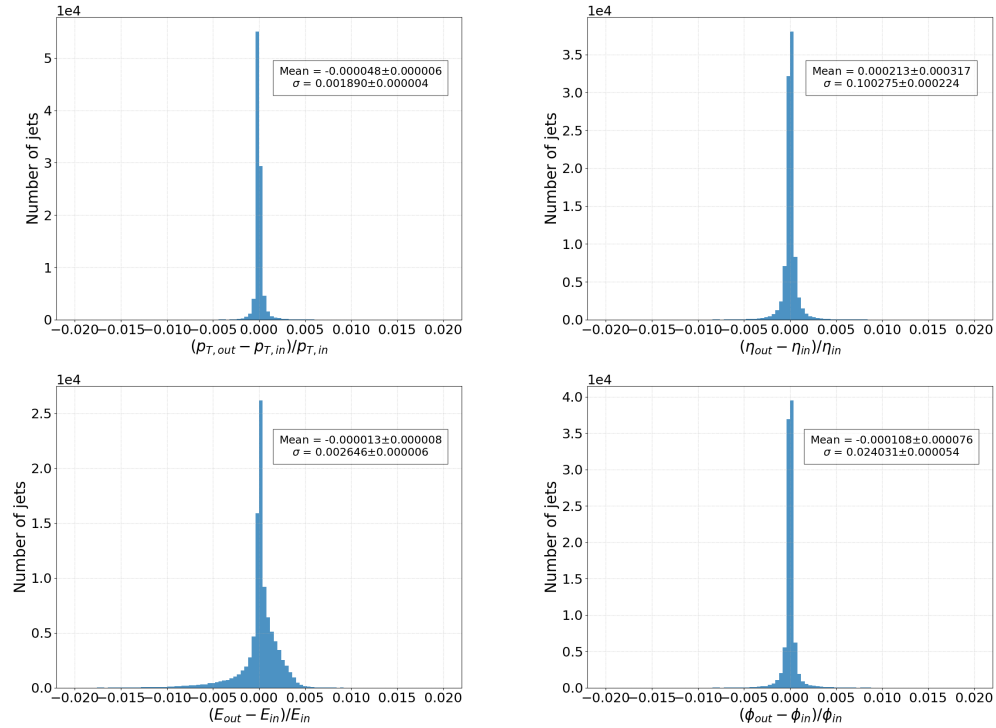

Figure 2.6: Distributions of residuals using the 3D latent space AE. Most residuals are smaller than 1%.

2D histograms of the residuals, as in Fig. 2.7, provide information about the correlation between the errors of different variables.

Fig. 2.7 demands some explanation. The diagonal contains the 1D histograms of the residuals while the plots under the diagonal show 2D histograms of all pairwise combinations. The white area in the 2D histograms represent the 1-σ contour curve. Inside the white area additional contour curves are drawn. Outside the 1-σ level individual data points are plotted as black dots. The background colors of the 2D histograms correspond to the correlation between the residuals of each variable pair in that histogram. As can be seen in the figure, there is no, or very little, correlation between the errors except for between E and p_T. This is not surprising since it is well known that E and p_T are highly correlated. All 2D histograms show a similar star-like pattern with long narrow arms stretching out along the x- and y-axes. This indicates that the AE rarely reconstructs more than 1 variable poorly in a given jet.

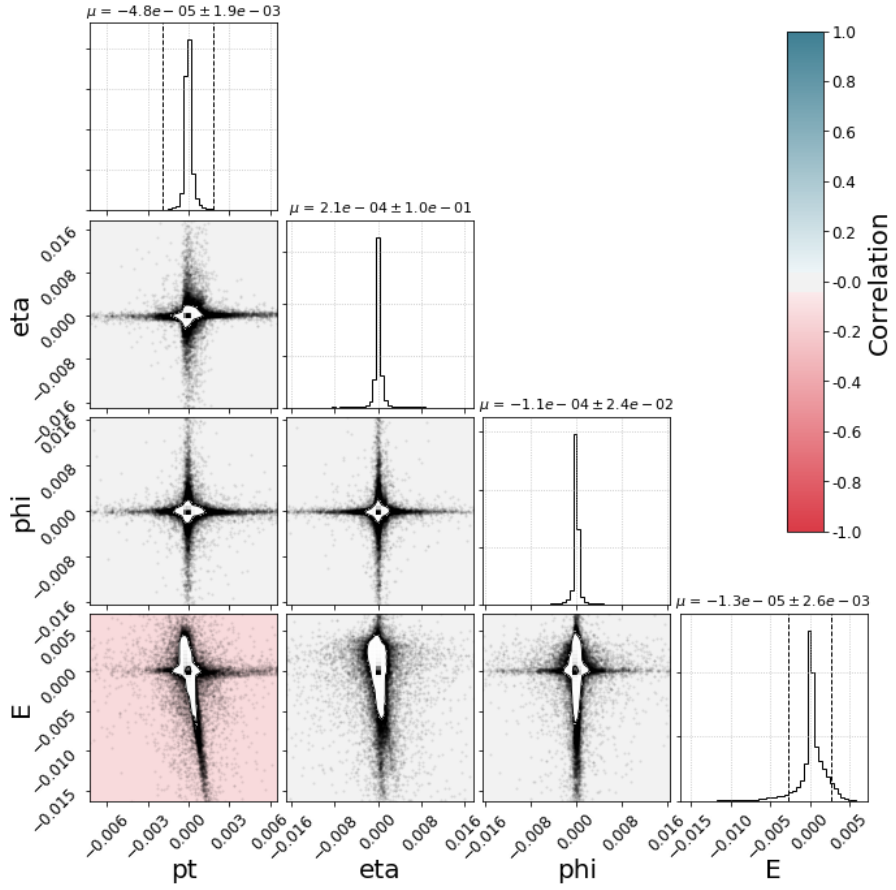

Figure 2.7: 2D histograms provide information about correlation between residuals. The background color of the 2D histograms corresponds to the correlation between the residuals of each variable pair. Axes have been scaled to include 99.5% of all jets. The scaling was done such that the bottom 0.25% and the top 0.25% of jets are missing from the plots.

2.3 27 variable input Autoencoders

In this section the AE is expanded in order to test a more realistic use case where most of the variables used to define a jet (Sec. 1.1.3) are included. Certain variables which are

structured as nested lists or which are booleans were left for future work.

Increasing the number of input variables results in increased training time. As a consequence it was not practical to explore as wide a range of different architectures as with the smaller network. Therefore, based on experience from training the four input variable AEs, the number of hidden layers was fixed to 7 and the number of nodes in each layer was fixed to 200 except for the latent space layer which was varied in size from 8 to 20. This allowed for studying the dependence of the reconstruction accuracy on the compression level. A sketch of the chosen architecture is provided in Fig. 2.8.

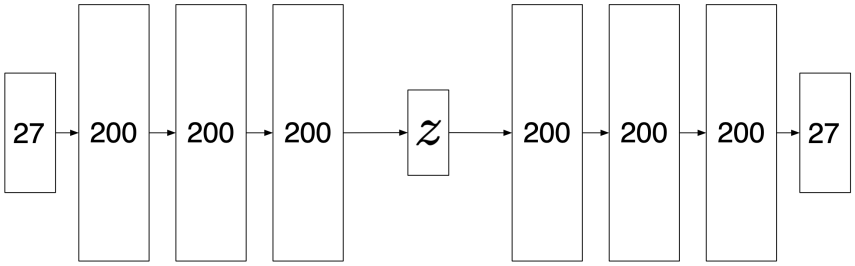

Figure 2.8: Sketch of the 27 variable input AE used in this project. The number of latent space dimensions, z, was varied between 8 and 20.

The increase in necessary computing power was solved by using 2 different HPC clusters, LUNARC's Aurora and HPC2N's Kebnekaise which will be described in more detail in Sec. 2.4. Using these, hyperparameter grid searches were performed, varying the batch size, learning rate (Sec. 1.2.2), weight decay parameter (Sec. 1.2.4) and number of latent space dimensions. Other activation functions were explored, first the ReLU and thereafter the LeakyReLU (Sec. 1.2.1) which proved to work better than the hyperbolic tangent when used in combination with batch normalization.

Furthermore, the 1cycle policy, introduced in [30], was explored and proved to result in better performance compared to a fixed learning rate using the same amount of training. The 1cycle policy suggests how to vary the learning rate and momentum over the course of training, starting out with a small learning rate, gradually increasing it to a maximum and then gradually decreasing it back to a small value. It was observed in [30] that the high learning rates during the middle part of the training process acted as a regularization method, preventing the model to get stuck in steep areas of the optimization landscape, preferring to land in flatter areas. In theory this would improve generalization since moving in any one direction on a flat area would not increase the value off the loss function very much. This is explained in [49] by using approximations of the hessian matrix, which describes the curvature in all directions in a high dimensional space. The authors of [49] show that approximations of the hessian matrix were lower when using the 1cycle policy than otherwise, indicating that a flatter area had been found.

Fig. 2.9 shows the learning rate and momentum schedules for the 1cycle policy as implemented in the fastai library. This differs slightly from [30] where the learning rate and momentum increase and decrease linearly with the number of training iterations but follows the same general idea.

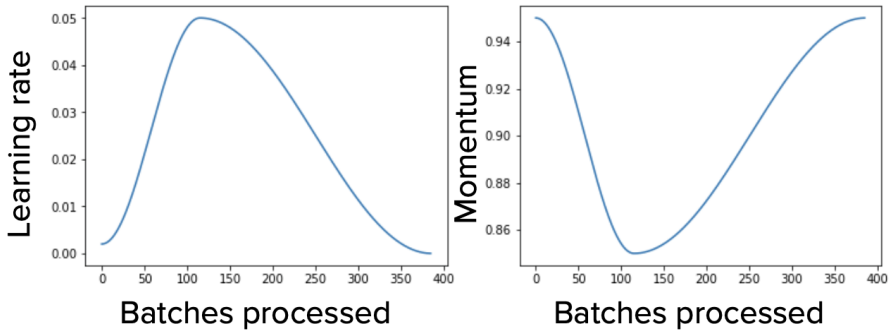

Figure 2.9: The learning rate and momentum schedule of the 1cycle policy as implemented in the fastai library.

2.3.1 The data

As for the 4 input variable AE, the data needs to be normalized. However, the 27 variables used here are so different in terms of their distribution over numerical values that the usual standardization does not render the variables to be of the same order of magnitude. Therefore, each individual variable was given a custom normalization designed for this purpose. A Python function was written to easily perform this normalization scheme as well as to do the inverse transformation in order to be able to unnormalize the variables. These Python functions can be found in [46]. The original distributions of all 27 variables are presented in Figs. 2.10, 2.11 and 2.12. Plots of the variable distributions after normalization can be found in Appendix A.5.

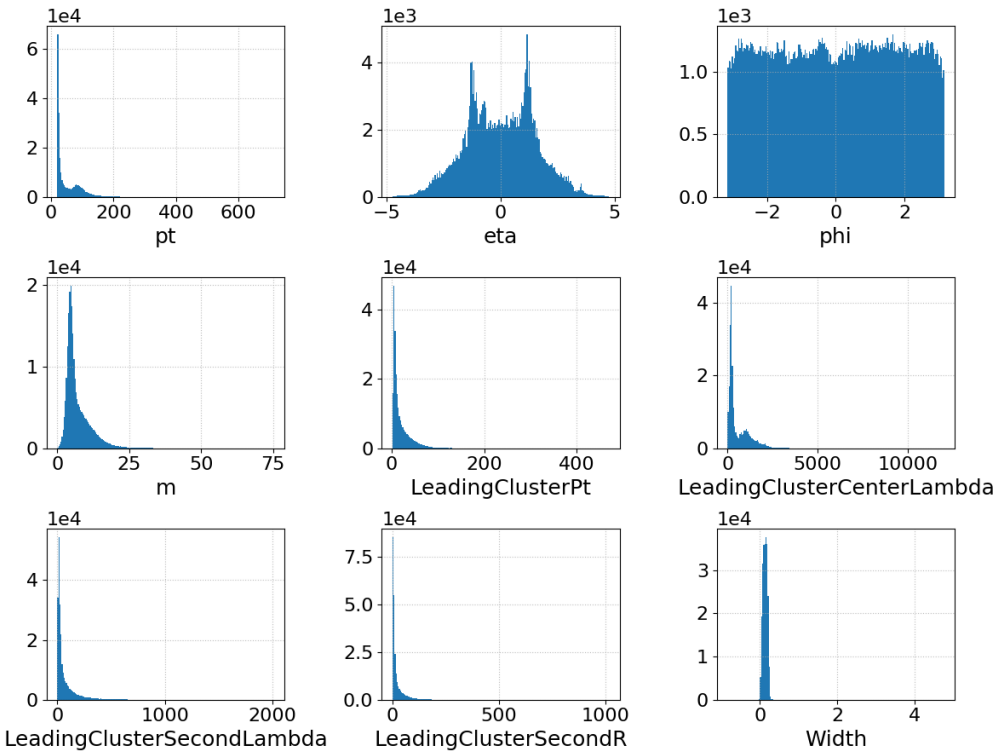

Figure 2.10: Distributions of a subset of the 27 variables used as input

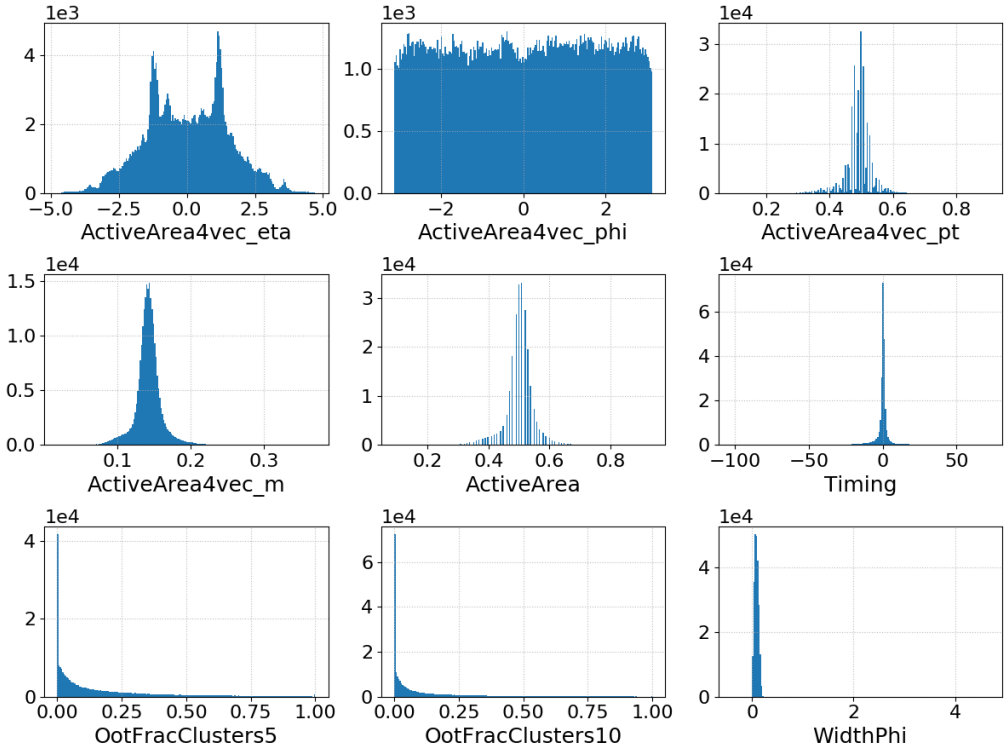

Figure 2.11: Distributions of a subset of the 27 variables used as input

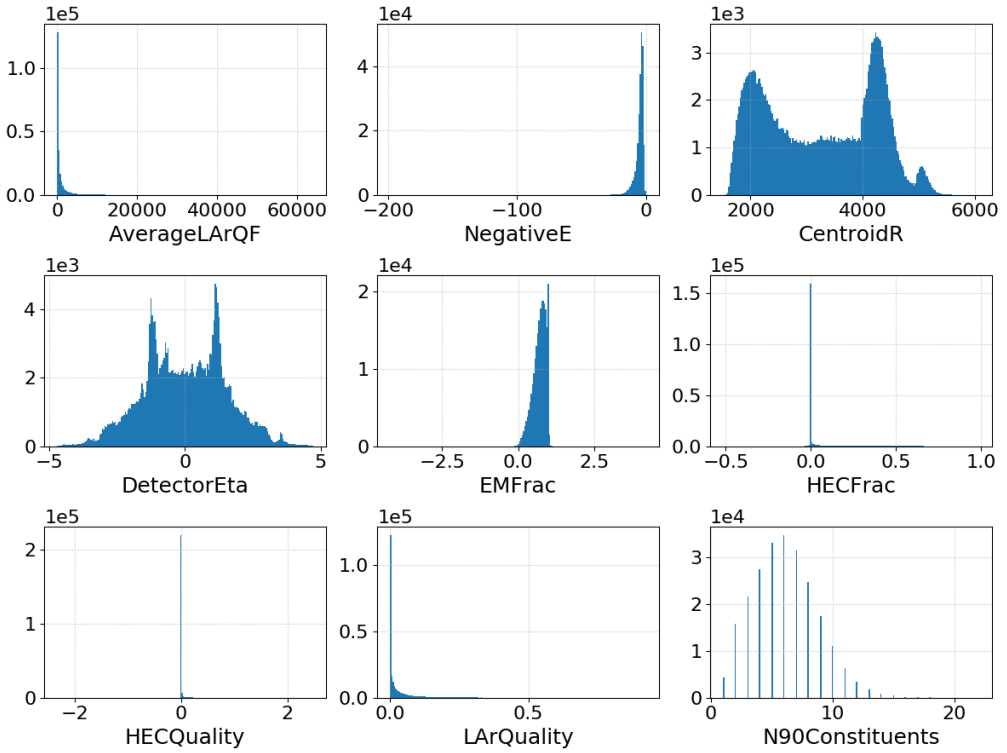

Figure 2.12: Distributions of a subset of the 27 variables used as input

2.3.2 Results and discussion

Hyperparameter grid searches

Hyperparameter grid searches were performed on LUNARC's Aurora and HPC2N's Kebnekaise. These resources and their use in this project will be described in more detail in Sec. 2.4. Due to limited resources a fairly coarse grid was searched. The choice of grid points was based on previous experience training the AEs "by hand". The grid searches were performed for all the different AEs (with different latent space dimensions), but in Fig. 2.13 the results from the 18D latent space AE are shown. There cannot be seen any simple linear dependence on the performance of the model on any of the hyperparameters varied in this search. They all seem to interact and the effect of changing one of them generally depends on the values of the other hyperparameters. The best performing network resulting from the search was trained using a learning rate of 10^{-2}, `wd` $= 0.01$ and `bs` $= 4096$.

Tabs. 2.3 and 2.4 summarize the best and median performing networks in terms of validation loss that resulted from the grid searches. As can be seen in Tab. 2.3 the best batch size always turned out to be 4096 while the best learning rate and weight decay parameter varied depending on the network used. In addition, it is seen that the validation loss depends monotonically on the size of the latent space of the AE. This dependence will be discussed further below.

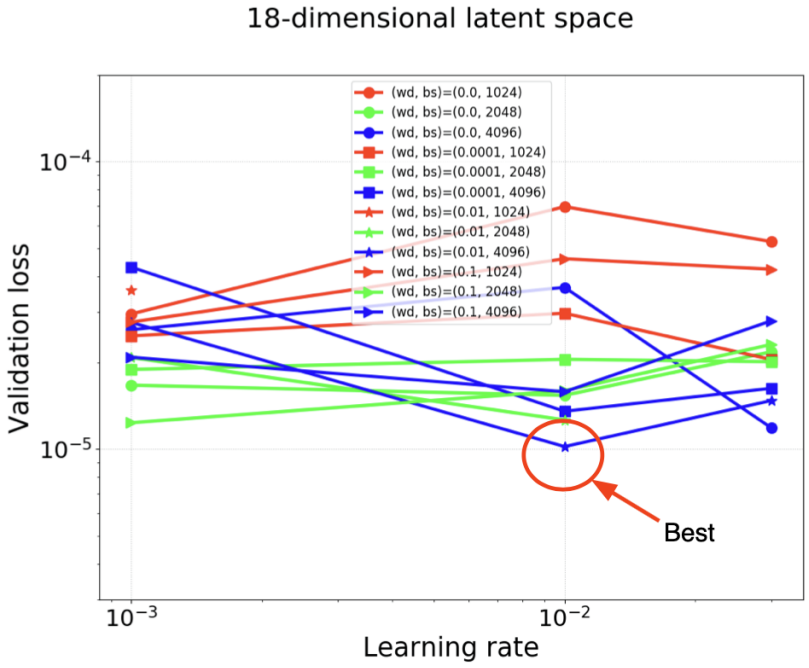

Figure 2.13: Summary of the results from a hyperparameter grid search using the 18D latent space AE with 27 input variables. Each color and marker corresponds to a different batch size (`bs`) and weight decay parameter (`wd`) as indicated in the legend. The best performing network is marked by a red circle and was trained using a learning rate of 10^{-2}, `wd` $= 0.01$ and `bs` $= 4096$.

A concern that the stochastic nature of NN training could result in different model performance when the same NN was trained more than once using the same hyperparameters led to the following test. The best performing NN from the grid search shown in Fig. 2.13 was retrained five times using the exact same hyperparameters whereafter the validation

Table 2.3: Best performing networks after grid search.

Latent space dimensions	Batch size	Learning rate	Weight decay	Validation loss	Training time
8	4096	1e-02	1e-04	4.712e-04	2 days, 0:00:45
10	4096	1e-03	1e-02	2.478e-04	2 days, 1:13:01
12	4096	1e-03	1e-01	1.241e-04	2 days, 0:53:15
14	4096	1e-02	1e-02	5.968e-05	2 days, 0:54:56
16	4096	3e-02	1e-04	2.679e-05	2 days, 0:37:20
18	4096	1e-02	1e-02	1.021e-05	2 days, 1:22:06
20	4096	1e-02	1e-02	5.649e-06	1 day, 22:27:46

Table 2.4: Median performing networks after grid search.

Latent space dimensions	Batch size	Learning rate	Weight decay	Validation loss	Training time
8	4096	1e-02	1e-02	4.982e-04	2 days, 0:04:48
10	4096	3e-02	1e-02	2.746e-04	2 days, 0:19:02
12	8192	3e-02	0	1.371e-04	2 days, 0:14:42
14	1024	1e-03	1e-02	7.354e-05	1 day, 21:02:45
16	4096	1e-03	0	3.644e-05	2 days, 1:50:36
18	2048	1e-03	1e-02	2.084e-05	1 day, 22:32:15
20	4096	3e-02	1e-04	1.661e-05	1 day, 23:59:55

loss was compared. Four of these ended up with approximately the same validation loss as before but one ended up with twice as high validation loss. The low number of data points in this experiment (5) means that it is not possible to accurately estimate how often this happens. Nevertheless, in order to investigate further, the training and validation losses as a function of training iterations were investigated more closely. In Fig. 2.14 the results from this investigation is shown and it can be seen there that the worse performing NN's validation loss suddenly starts to increase at the end of training. This could be because the optimization algorithm happened to find a particular local minimum in the optimization landscape that does not generalize well. It should be noted that the trainable weights that resulted in the lowest validation loss during the entire length of the training process are the weights that get saved. So in the case of the left plot in Fig. 2.14, the validation loss of the resulting model is not the one at the very right end of the curve but rather that of the lowest point on that curve.

In conclusion, when running grid searches it is a good idea to run them more than once if time is not a limiting factor. Since it often is though, a look at the training and validation loss curves could give some hints as to whether the model has reached a good local minimum or not.

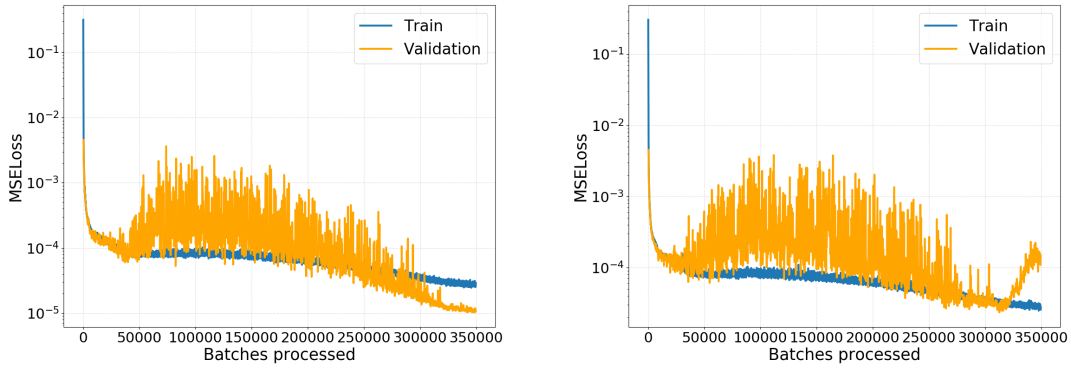

Figure 2.14: Comparison of two training runs of the 18D latent space AE using the same hyperparameters as the best performing network from the grid search in Fig. 2.13. (Left) The results from a NN that achieved the same end performance as the NN from the grid search. (Right) The network that resulted in roughly twice as high validation loss.

Reconstructed variable distributions

Figs. 2.15 and 2.16 compares the distributions of all variables before and after they have been compressed and decompressed using the 14D latent space AE. The agreement between the input and output is not as good as for the 4 variable input AE but considering the compression level is higher (compressing from 27 to 14 variables instead of 4 to 3) the distributions still agree well.

Some variables have discrete distributions, namely `ActiveArea` and `N90Constituents`. These were not treated differently in any way during the training of the AEs but after training a slight modification was used to improve accuracy of the reconstruction. The output of the AE was rounded to the closest discrete value present in the input distribution of the variable. Although there are in principle better methods to handle discrete variables, like one-hot encoding for instance, the rounding produces great results, especially in the case of `N90Constituents` where the reconstruction is close to perfect.

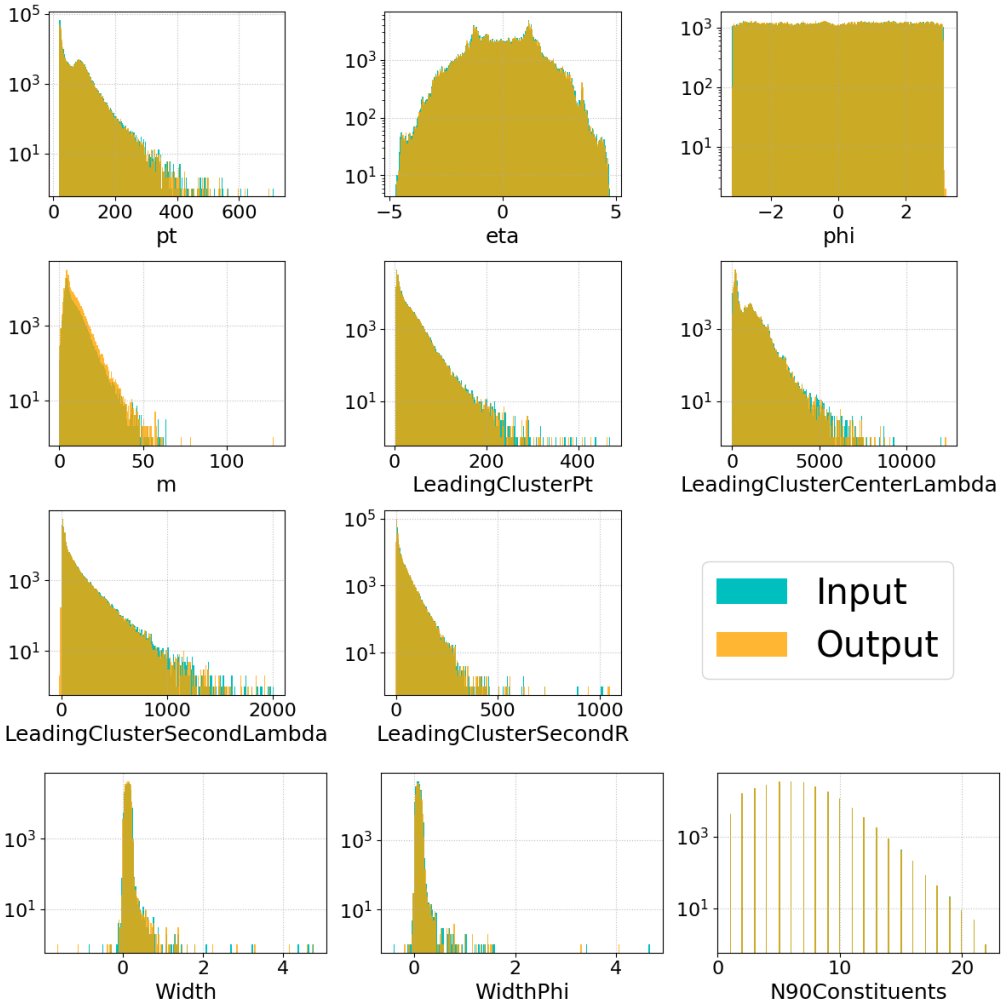

Figure 2.15: Variable distributions of the 14D latent space AE output on top of input.

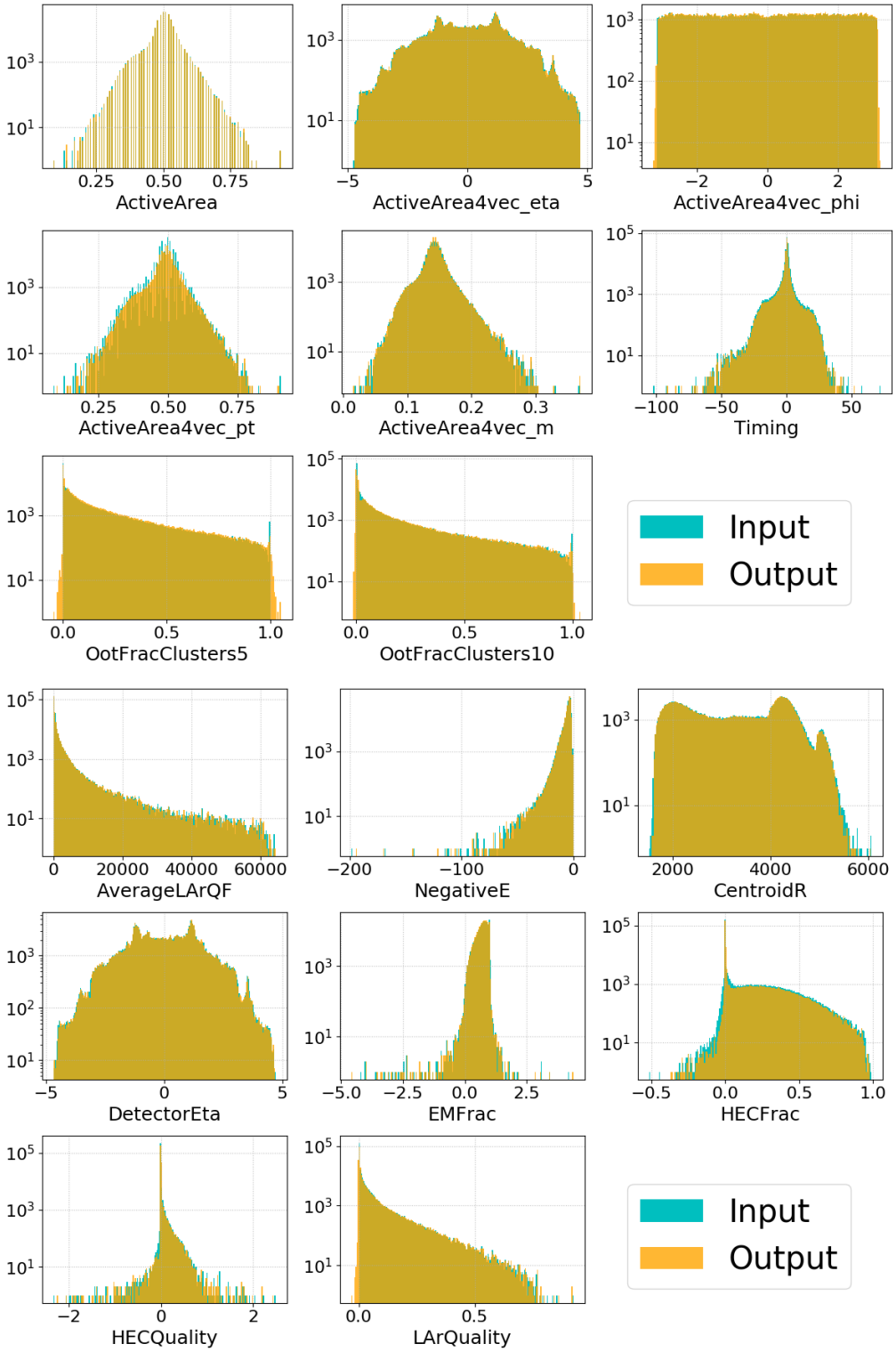

Figure 2.16: Variable distributions of the 14D latent space AE output on top of input.

Residuals

The residuals in Fig. 2.17 are, not surprisingly, larger than for the 4 variable input AE. There is now a strong correlation between the error of the jet mass m, and the transverse momentum p_T indicating that the network has learned to use the correlation between those variables to compress the data. Note that for η and ϕ it is the absolute difference that is plotted, not the relative.

Residuals for the remaining variables not presented in Fig. 2.17 can be found in Appendix A.4.

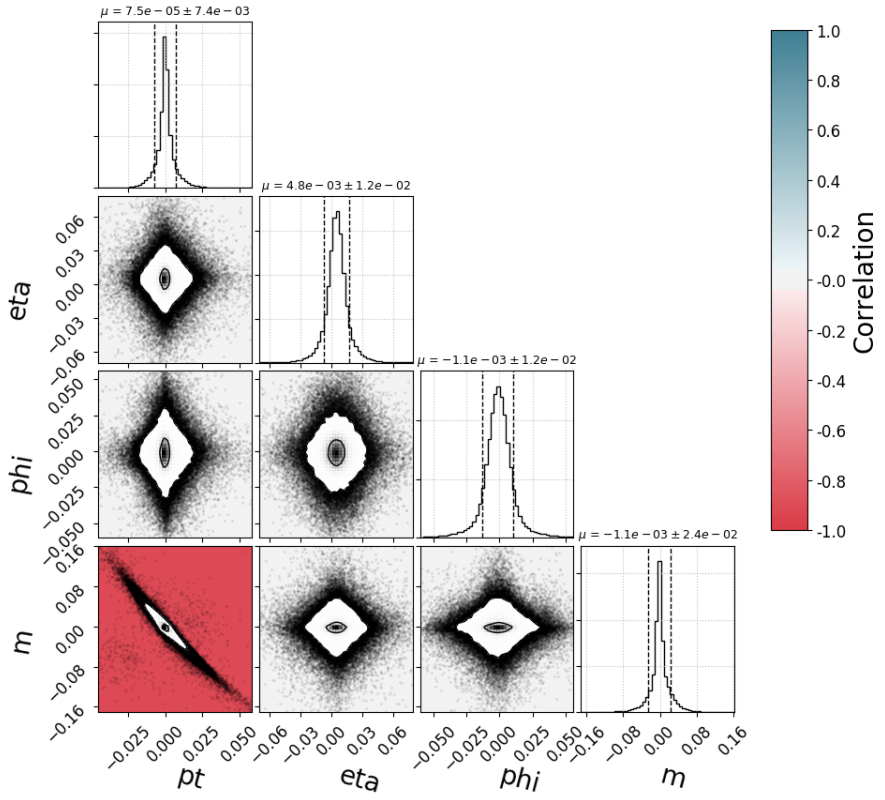

Figure 2.17: Residuals of the 18D latent space AE. Values shown are relative difference for m and p_T, absolute difference for η and ϕ. Axes have been scaled to include 99.9% of all jets.

Performance dependence on compression level

The plots in Fig. 2.18 show the best and median performing AEs in terms of validation loss that were trained during grid searches. In the plots, the mean and standard deviation of the residuals are plotted as a function of the number of latent space dimensions. A good AE should render these metrics as close to zero as possible. As expected, there is a clear correlation between compression level (number of latent space dimensions) and the performance, higher compression leads to worse reconstruction accuracy.

Fig. 2.19 and Tab. 2.5 show the dependence of the size of the latent space on the average time required for encoding and decoding data, normalized to the number of jets in the dataset. There is a general trend that a larger latent space results in longer encoding and decoding time. This is not always the case though as, for instance, the encoding was faster using 16 latent space dimensions compared to using 10, 12 or 14. Nevertheless the

differences are small and could depend on fluctuations in CPU performance. The averages were computed from 100 encoding and decoding operations for each AE.

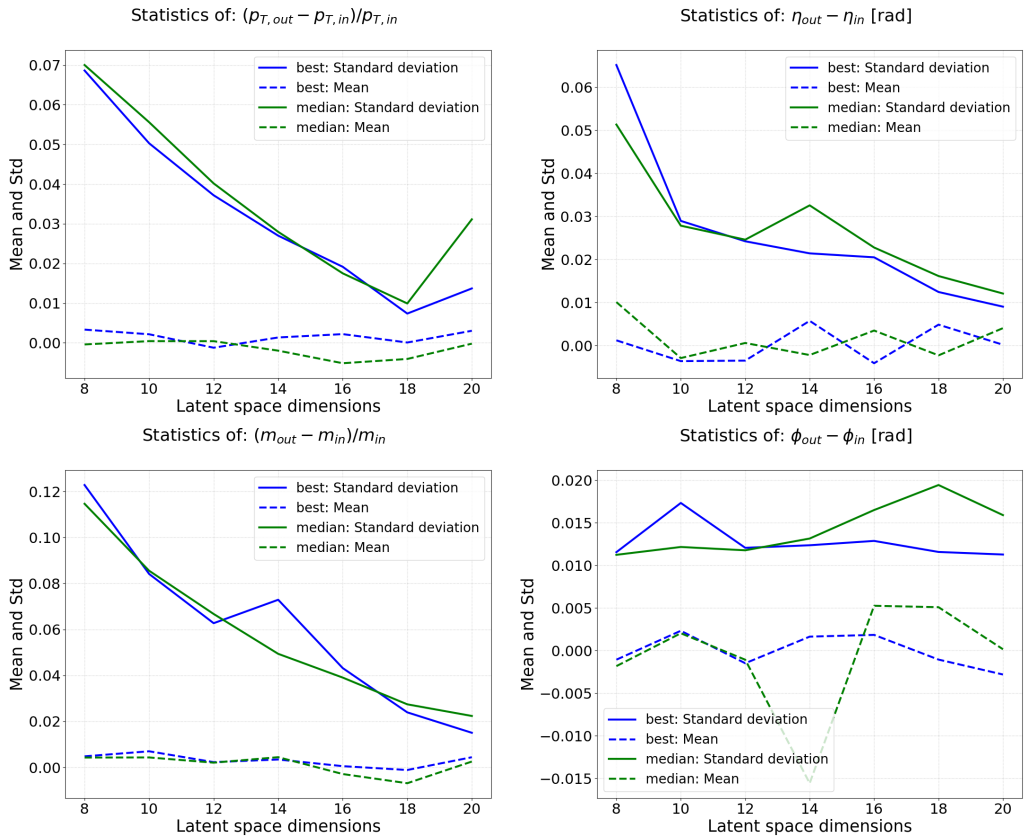

Figure 2.18: Performance of the AE in terms of standard deviation and mean of the residuals. Values shown are relative difference for m and p_T, absolute difference for η and ϕ.

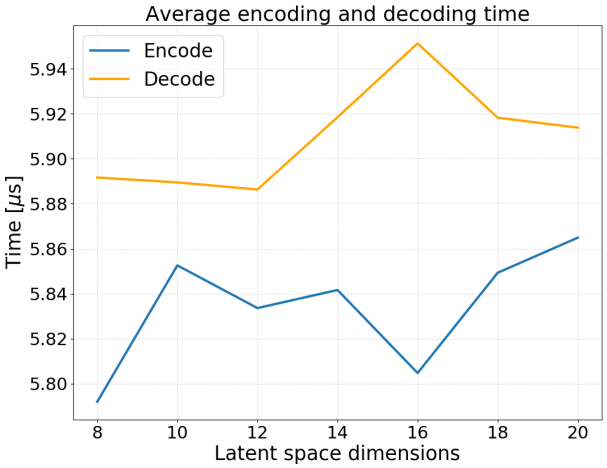

Figure 2.19: The average time per jet required for encoding and decoding for AEs with different latent space sizes. The average was computed from 100 encoding and decoding operations on the entire validation set of $\mathcal{D}_C$ on a 2013 MacBook Pro with a 2.6 GHz Intel Core i5 CPU.

Table 2.5: The average time per jet required for encoding and decoding for AEs with different latent space sizes. The average was computed from 100 encoding and decoding operations on the entire validation set of $\mathcal{D}_C$ on a 2013 MacBook Pro with a 2.6 GHz Intel Core i5 CPU.

Latent space dimensions	Encoding time [μs/jet]	Decoding time [μs/jet]
8	5.79	5.89
10	5.85	5.89
12	5.83	5.89
14	5.84	5.92
16	5.80	5.95
18	5.85	5.92
20	5.86	5.91

Reconstructing a dijet mass signal

To further evaluate the capabilities of the AEs, a signal MC sample of a DM mediator decaying into dijets was used. The reason for this choice is that many searches for resonant particles decaying into dijets rely on the appearance of a mass peak over a smooth background in the dijet mass distribution. Therefore it is important that the AEs are able to accurately reconstruct such mass peaks.

To test this, the jet data of the MC sample was compressed and reconstructed using the best performing AEs whereafter the dijet mass distribution was computed by using the two highest p_T jets in each collision event and Eq. 1.3.

Looking at Fig. 2.20, showing the 12D latent space AE's ability to reconstruct a signal coming from a DM mediator with a mass of 350 GeV, it can be seen that the mean and standard deviation of the signal peak is reconstructed well, with differences within the dijet mass resolution [50]. This is a promising sign for the future use of this network, since the AE had not been trained on signals like this but only on experimental background data.

Plots like the ones in Figs. 2.20 and 2.21 for AEs with a latent space size of $z \in \{8, 10, 12, 14, 16, 18, 20\}$ can be found in Appendix A.3.

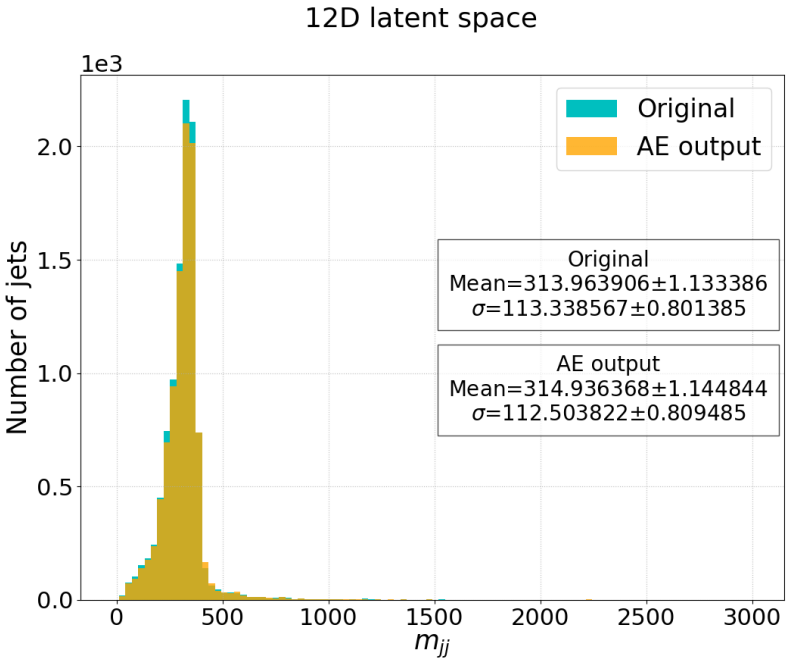

Figure 2.20: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$) with a truth-level mass of 350 GeV. The data was compressed and decompressed by the 12D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

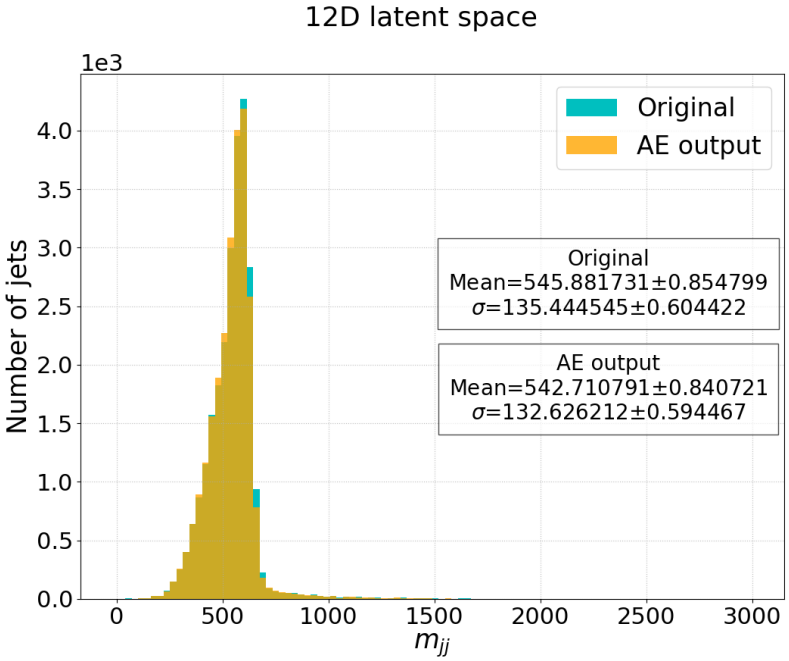

Figure 2.21: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$) with a truth-level mass of 600 GeV. The data was compressed and decompressed by the 12D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

2.4 Computing resources

Several different computing resources were experimented with during training of the AEs used in this work. The initial prototyping of network architectures was done on the CPU of a laptop computer, a 2013 MacBook Pro with an Intel Core i5 CPU and 16 GB of memory. For small NNs and small datasets ($<$ 1 million jets) training and evaluation was smooth. When exploring larger NNs with 5 or more layers and 50 or more nodes in each layer experimentation became slow and time consuming.

In a second stage, after the initial prototyping, a wider range of architectures with varying numbers of layers and nodes were investigated. This required more computing power and was done on *Kebnekaise*, the largest supercomputer at the High Performance Computing Center North (HPC2N). Kebnekaise provides almost 50 GPU equipped nodes and over 500 CPU nodes with a total of over 17000 cores. HPC2N is part of The Swedish National Infrastructure for Computing (SNIC) which is an organization that makes available large scale HPC resources for Swedish researchers. Although NNs often can be trained much faster on GPUs compared to CPUs the much lower number of available GPUs resulted in jobs having to queue for longer before running. The impact of the queuing on the total amount of time from a batch of jobs being submitted to all of them being completed were such that it proved faster to use CPUs. The average queuing time for CPU usage was much shorter and there were no difficulties in using many of them at once. No quantitative analysis of the time required to run jobs on GPU nodes versus CPU nodes was done. Nevertheless, when using GPUs only a small fraction of the submitted jobs in the grid search would typically have started running after 5 hours whereas all or almost all of the jobs would already have both started and finished by then when using CPU nodes. The GPU nodes could still be useful, for instance when running a small number of compute intensive jobs since then the proportion of total time that is spent on queuing is smaller.

When the need for more data arose, the supercomputer *Aurora* at the center for scientific and technical computing at Lund University (LUNARC) was used. The reason being that data suitable for this thesis was stored on disks connected to Aurora. LUNARC's Aurora is somewhat smaller than Kebnekaise with 180 compute nodes offering 20 cores per node for SNIC use and 50 nodes for dedicated Lund University research projects.

Chapter 3

Conclusions

3.1 Conclusions

AEs have been designed for, and trained to, encode and decode data containing hadronic jets reconstructed at the trigger level from the ATLAS detector at CERN. Initially, a simple AE reducing 4 input variables to a 3-dimensional latent space was developed as a proof of concept whereafter the number of inputs was increased to 27.

The performance of trained AEs were measured not only in terms of the value of the loss function, the MSE, but also in terms of the difference, and relative difference, between the input and output of the AEs.

The AEs were shown to successfully compress and reconstruct simple jet data. From preliminary tests on data and signal the quality of the reconstruction is good enough for applications such as TLA where a high fidelity is not required since the search using these data is robust against reasonable performance deterioration. The performance of the AEs were successfully validated on a signal MC sample not seen by the AEs during training.

3.2 Possible improvements and future work

Some of the 27 variables used as input in the second stage of the project had discrete distributions but were not treated any differently than the continuous variables, apart from that the output was rounded to the closest value present in the discrete input. Using a more appropriate technique to handle the discrete variables, e.g. one-hot encoding, would probably improve reconstruction accuracy.

Exploring a wider variety of different NN architectures, for instance skipping connections, auxiliary inputs, multiple branches, more/fewer nodes or more/fewer layers could possibly improve results further.

An important thing to investigate is the likelihood of the AEs to make spurious jets (noise) appear as normal jets and vice versa since that would hurt reconstruction accuracy.

Another interesting point to explore is to chain the AE's dimensionality reduction with a standard, lossless, compression algorithm used by ATLAS in order to compute the total compression factor. It is not generally true that chaining compression algorithms together will result in them achieving the same compression ratio as they would individually. For instance, it is possible to conceive of the AE having used patterns present in the data in order to compress it which then could leave the lossless compression algorithms with less patterns to use.

Last but not least, the time needed for compression and decompression could be evaluated further on hardware that would realistically be used in operation. The tests performed in this work were, as mentioned, performed on a laptop and hence is only an indication of the difference in required computation relative the various AEs and not a good indication of the absolute time that would be required on other pieces of hardware.

References

[1] Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B* **716**, 1 – 29 (2012). URL http://www.sciencedirect.com/science/article/pii/S037026931200857X.

[2] Berners-Lee, T. Information management: A proposal. http://cds.cern.ch/record/369245/files/dd-89-001.pdf (1989).

[3] Einstein, A. Ist die trägheit eines körpers von seinem energieinhalt abhängig? *Annalen der Physik* **323**, 639–641 (1905). URL https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19053231314. https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.19053231314.

[4] Cush. https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg.

[5] Thomson, M. *Modern Particle Physics* (Cambridge University Press, 2013). ISBN: 9781107034266.

[6] Suttons, C. & Zervas, P. The W and Z at LEP. *CERN Courier* (2004). https://cerncourier.com/a/the-w-and-z-at-lep/.

[7] Clowe, D. *et al.* A direct empirical proof of the existence of dark matter. *The Astrophysical Journal* **648**, L109–L113 (2006). URL https://doi.org/10.1086%2F508162.

[8] Roos, M. Dark Matter: The evidence from astronomy, astrophysics and cosmology. *arXiv e-prints* arXiv:1001.0316 (2010). 1001.0316.

[9] Hinshaw, G. *et al.* Five-Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Data Processing, Sky Maps, and Basic Results. *Astrophys. J. Suppl.* **180**, 225–245 (2009). 0803.0732.

[10] Bertone, G. & Hooper, D. A History of Dark Matter. *Rev. Mod. Phys.* **90**, 045002 (2018). 1605.04909.

[11] Quark confinement. *Lokal_profil*, CC-BY-SA-2.5.

[12] The ATLAS Collaboration *et al.* The ATLAS experiment at the CERN large hadron collider. *Journal of Instrumentation* **3**, S08003–S08003 (2008). URL https://doi.org/10.1088%2F1748-0221%2F3%2F08%2Fs08003.

[13] Trigger and data acquisition system. https://atlas.cern/discover/detector/trigger-daq (2019).

[14] Aaij, R. *et al.* Tesla : an application for real-time data analysis in High Energy Physics. *Comput. Phys. Commun.* **208**, 35–42 (2016). 1604.05596.

[15] Sirunyan, A. M. *et al.* Search for dijet resonances in proton–proton collisions at $\sqrt{s} =$ 13 TeV and constraints on dark matter and other models. *Phys. Lett.* **B769**, 520–542 (2017). [Erratum: Phys. Lett.B772,882(2017)], 1611.03568.

[16] Aaboud *et al.*, M. Search for low-mass dijet resonances using trigger-level jets with the atlas detector in *pp* collisions at $\sqrt{s} = 13$ TeV. *Phys. Rev. Lett.* **121**, 081801 (2018). URL https://link.aps.org/doi/10.1103/PhysRevLett.121.081801.

[17] Chala, M., Kahlhoefer, F., McCullough, M., Nardini, G. & Schmidt-Hoberg, K. Constraining dark sectors with monojets and dijets. *Journal of High Energy Physics* **2015**, 89 (2015). URL https://doi.org/10.1007/JHEP07(2015)089.

[18] The ATLAS Collaboration. https://twiki.cern.ch/twiki/pub/AtlasPublic/ComputingandSoftwarePublicResults/diskHLLHC_noold.png (2019).

[19] Dahl, G. E., Jaitly, N. & Salakhutdinov, R. Multi-task Neural Networks for QSAR Predictions. *arXiv e-prints* arXiv:1406.1231 (2014). 1406.1231.

[20] Litjens, G. *et al.* A survey on deep learning in medical image analysis. *Medical Image Analysis* **42**, 60 – 88 (2017). URL http://www.sciencedirect.com/science/article/pii/S1361841517301135.

[21] Baldi, P., Sadowski, P. & Whiteson, D. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Commun.* **5**, 4308 (2014). 1402.4735.

[22] Rosenblatt, F. F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* **65 6**, 386–408 (1958).

[23] Hochreiter, S., Bengio, Y., Frasconi, P. & Schmidhuber, J. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001).

[24] Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986). doi:10.1038/323533a0.

[25] Douglas, S. C. & Yu, J. Why relu units sometimes die: Analysis of single-unit error backpropagation in neural networks. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 864–868 (2018).

[26] Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016). http://www.deeplearningbook.org.

[27] Bottou, L. Large-scale machine learning with stochastic gradient descent. *Proc. of COMPSTAT* (2010).

[28] Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning*, p. 292–296 (MIT Press, 2016). http://www.deeplearningbook.org.

[29] Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv e-prints* arXiv:1412.6980 (2014). 1412.6980.

[30] Smith, L. N. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR* **abs/1803.09820** (2018). URL http://arxiv.org/abs/1803.09820. 1803.09820.

[31] Smith, L. N. No more pesky learning rate guessing games. *CoRR* **abs/1506.01186** (2015). URL http://arxiv.org/abs/1506.01186. 1506.01186.

[32] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR* **abs/1207.0580** (2012). URL http://arxiv.org/abs/1207.0580. 1207.0580.

[33] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**, 1929–1958 (2014). URL http://jmlr.org/papers/v15/srivastava14a.html.

[34] Loshchilov, I. & Hutter, F. Fixing weight decay regularization in adam. *CoRR* **abs/1711.05101** (2017). URL http://arxiv.org/abs/1711.05101. 1711.05101.

[35] Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR* **abs/1502.03167** (2015). URL http://arxiv.org/abs/1502.03167. 1502.03167.

[36] Bjorck, J., Gomes, C. P. & Selman, B. Understanding batch normalization. *CoRR* **abs/1806.02375** (2018). URL http://arxiv.org/abs/1806.02375. 1806.02375.

[37] Santurkar, S., Tsipras, D., Ilyas, A. & Madry, A. How Does Batch Normalization Help Optimization? *arXiv e-prints* arXiv:1805.11604 (2018). URL https://arxiv.org/abs/1805.11604. 1805.11604.

[38] Ellison, D. Fraud Detection Using Autoencoders in Keras with a TensorFlow Backend. https://www.datascience.com/blog/fraud-detection-with-tensorflow (2018).

[39] Hinton, G. & Salakhutdinov, R. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)* **313**, 504–7 (2006).

[40] Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, 1096–1103 (ACM, New York, NY, USA, 2008). URL http://doi.acm.org/10.1145/1390156.1390294.

[41] Ladjal, S., Newson, A. & Pham, C. A pca-like autoencoder. *CoRR* **abs/1904.01277** (2019). URL http://arxiv.org/abs/1904.01277. 1904.01277.

[42] Williams, M. & Weisser, C. Personal communication.

[43] The ATLAS Collaboration. Jet energy measurement with the ATLAS detector in proton-proton collisions at $\sqrt{s} = 7$ TeV. *The European Physical Journal C* **73**, 2304 (2013). URL https://doi.org/10.1140/epjc/s10052-013-2304-2.

[44] Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. *et al.* (eds.) *Advances*

in Neural Information Processing Systems 32, 8024–8035 (Curran Associates, Inc., 2019). URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[45] Howard, J. *et al.* fastai. https://github.com/fastai/fastai (2018).

[46] Wulff, E. G. T. Hadron Jet Compression using Deep Autoencoders. https://github.com/erwulff/lth_thesis_project (2019).

[47] Gugger, S. The 1cycle policy. https://sgugger.github.io/the-1cycle-policy.html (2018).

[48] Petersen, T. Personal communication.

[49] Smith, L. N. & Topin, N. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR* **abs/1708.07120** (2017). URL http://arxiv.org/abs/1708.07120. 1708.07120.

[50] Aaboud, M. *et al.* Search for low-mass dijet resonances using trigger-level jets with the ATLAS detector in pp collisions at $\sqrt{s} = 13$ TeV. *Phys. Rev. Lett.* **121**, 081801 (2018). 1804.03496.

[51] Bryngemark, L. *Search for new phenomena in dijet angular distributions at √s = 8 and 13 TeV*. Ph.D. thesis, Lund U. (2016-02-23).

[52] Selection of jets produced in 13TeV proton-proton collisions with the ATLAS detector. Tech. Rep. ATLAS-CONF-2015-029, CERN, Geneva (2015). URL https://cds.cern.ch/record/2037702.

[53] Aad, G. *et al.* ATLAS Measurements of the Properties of Jets for Boosted Particle Searches. *Phys. Rev.* **D86**, 072006 (2012). 1206.5369.

[54] Aad, G. *et al.* Topological cell clustering in the ATLAS calorimeters and its performance in LHC Run 1. *Eur. Phys. J.* **C77**, 490 (2017). 1603.02934.

Appendix A

A.1 Dataset original file names

Table A.1: Dataset file names.

Name	Derived from file
$\mathcal{D}_A$	user.breynold.data16_13TeV.00307656.physics_Main.DAOD_NTUP_JTRIG_JETM1.r9264_p3083_p3601_j042_tree.root
$\mathcal{D}_B$	data18_13TeV.00364292.calibration_DataScouting_05_Jets.deriv.DAOD_TRIG6.r10657_p3592_p3754
$\mathcal{D}_{MC}$	mc16_13TeV.307791.MGPy8EG_N30LO_A14N23LO_DMsA_dijet_mR0p6_gSM0p05.deriv.DAOD_EXOT2.e5687_a875_r9364_p3654

A.2 Variable descriptions

- `pt` - Transverse momentum, p_T.
- `eta` - Jet pseudorapidity, η.
- `phi` - Azimuthal angle, ϕ.
- `m` - Mass of the jet.
- `ActiveArea` - Area of the jet, calculated using the overlap of the jet with so-called "ghost particles" [51].
- `ActiveArea4vec_eta` - The jet area can be calculated by the vector sum of the four-momenta of the ghost particles, and be a four-momentum itself.
- `ActiveArea4vec_m` - See above.
- `ActiveArea4vec_phi` - See above.
- `ActiveArea4vec_pt` - See above.
- `AverageLArQF` - The energy–squared weighted average of the pulse quality of the electromagnetic calorimeter cells, used for distinguishing fake jets from e.g. calorimeter noise from real jets (*jet cleaning*, see [52]).
- `NegativeE` - Negative energy created by sporadic noise in calorimeter cells, used for jet cleaning (see above).
- `HECQuality` - Pulse quality of the hadronic endcap calorimeter, used for jet cleaning (see above).

- `LArQuality` - Pulse quality of the LAr calorimeter cells, used for jet cleaning (see above).

- `Width` - p_T-weighted radial distance of the jet components with respect to the jet axis [53].

- `WidthPhi` - Same as above, but only distance in ϕ.

- `CentroidR` - Energy-weighted barycenter of the jet, used for jet cleaning.

- `DetectorEta` - Jet pseudorapidity, calculated from the geometrical center of the detector rather than from the collision vertex like `eta`.

- `LeadingClusterCenterLambda` - Cluster moments calculated from the leading cluster in the jet [54].

- `LeadingClusterPt` - See above.

- `LeadingClusterSecondLambda` - See above.

- `LeadingClusterSecondR` - See above.

- `N90Constituents` - Number of constituents of the jet containing 90% of the total jet energy.

- `EMFrac` - Fraction of the jet energy in the electromagnetic calorimeters.

- `HECFrac` - Fraction of the jet energy in the hadronic endcap calorimeter.

- `Timing` - Timing of the jet with respect to the LHC clock (where 0 is in-time with the collision), based on the energy-weighted timing of the clusters composing the jet.

- `OotFracClusters10` - Fraction of clusters with timing above 10 ns from the collision time.

- `OotFracClusters5` - Fraction of clusters with timing above 5 ns from the collision time.

A.3 Dijet mass signal peak

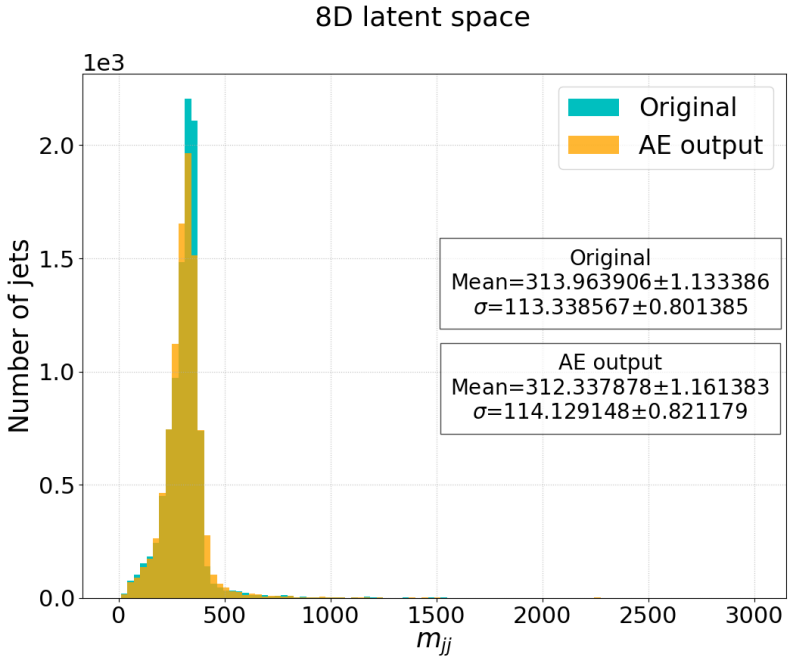

Figure A.1: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 8D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

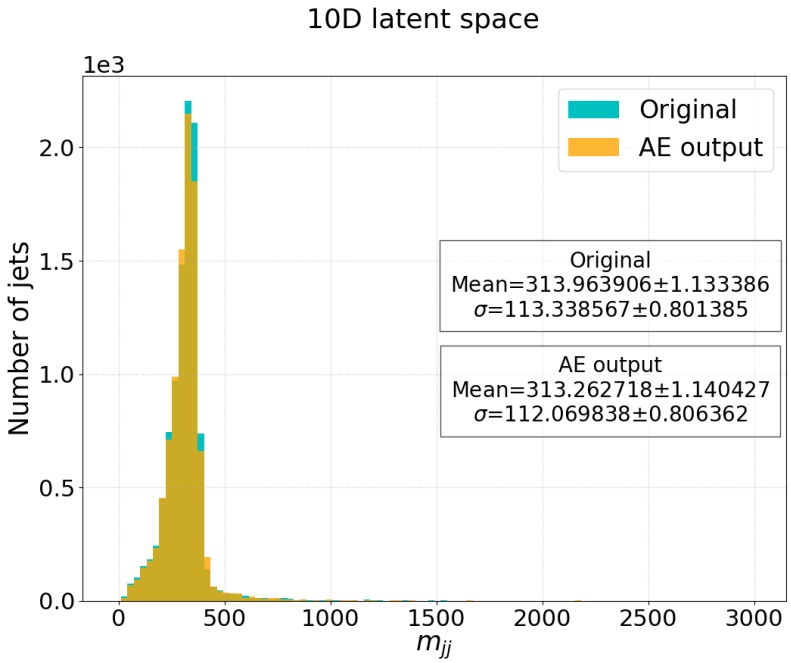

Figure A.2: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 10D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

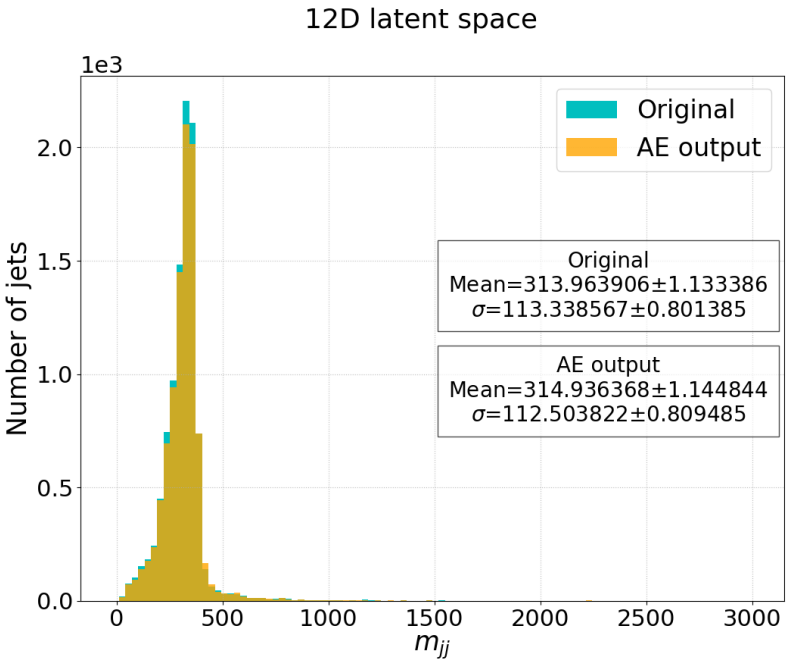

Figure A.3: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 12D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

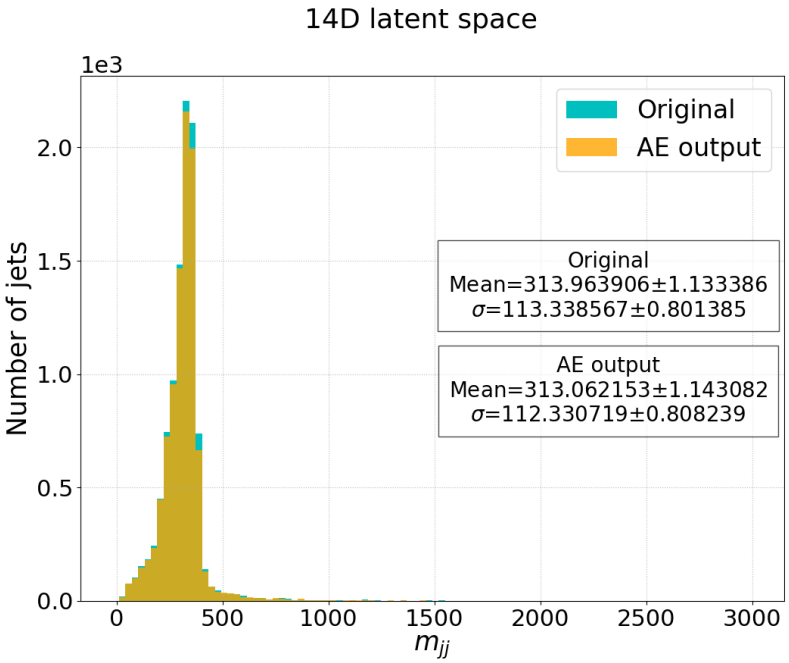

Figure A.4: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 14D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

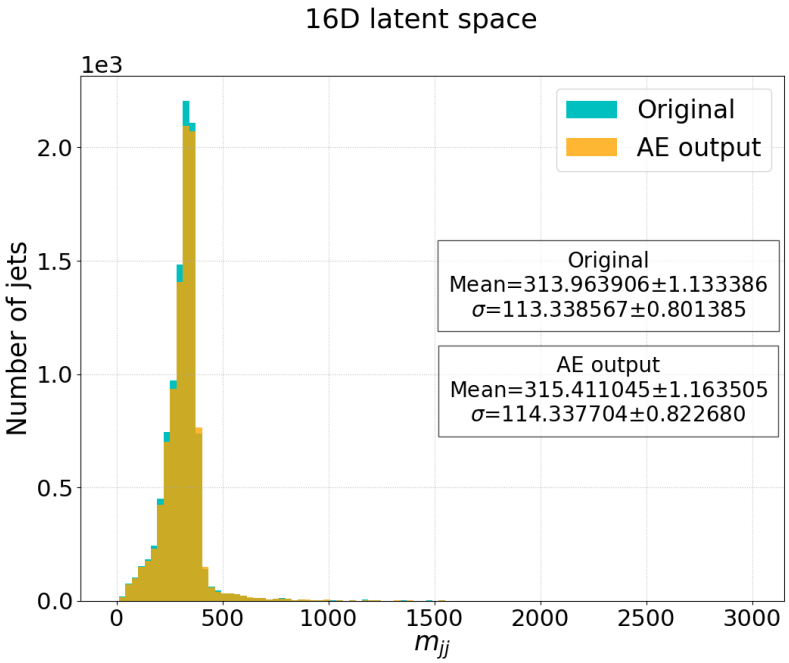

Figure A.5: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 16D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

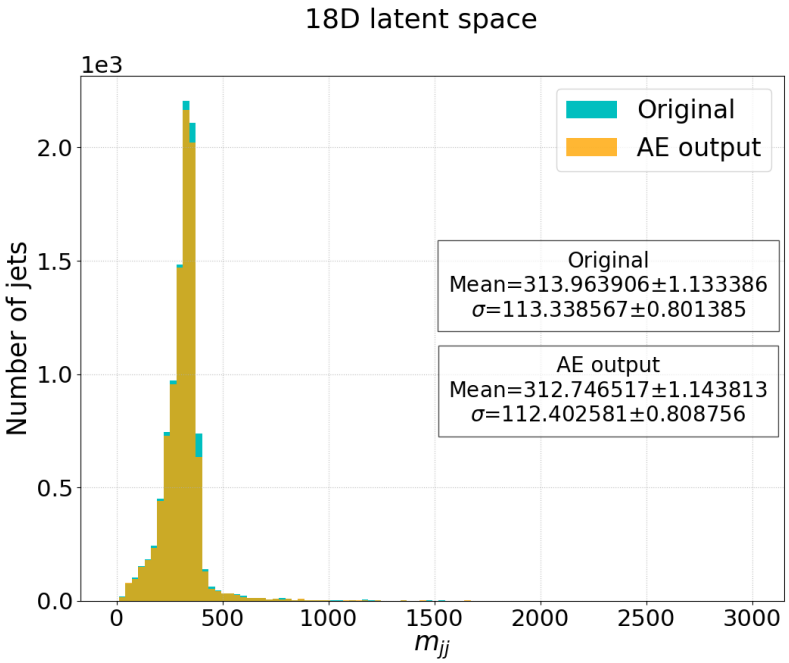

Figure A.6: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 18D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

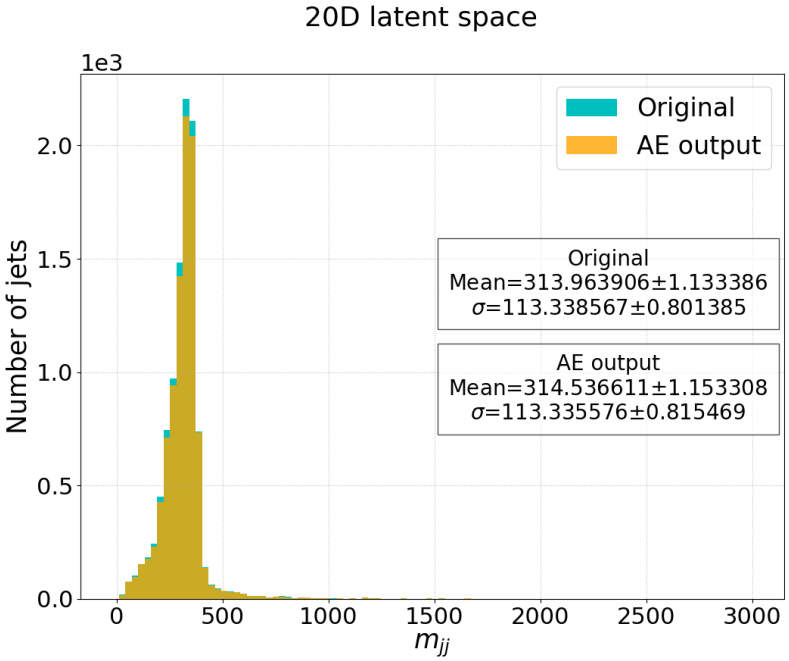

Figure A.7: Comparison of the dijet mass distribution from a signal MC sample ($\mathcal{D}_{MC}$). The data was compressed and decompressed by the 20D latent space AE whereafter the dijet mass was computed and compared here with the original dijet mass distribution.

A.4 Residuals of the 27 variable input AEs

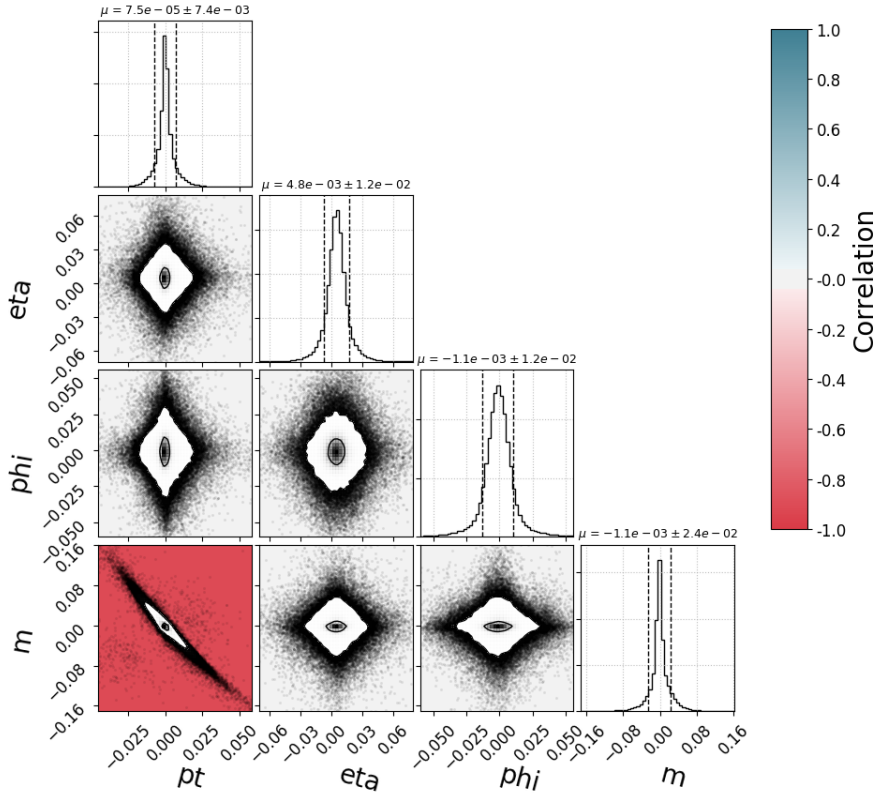

Figure A.8: Residuals of the 18D latent space AE. Values shown are relative difference for m and p_T, absolute difference for η and ϕ. Axes have been scaled to include 99.9% of all jets.

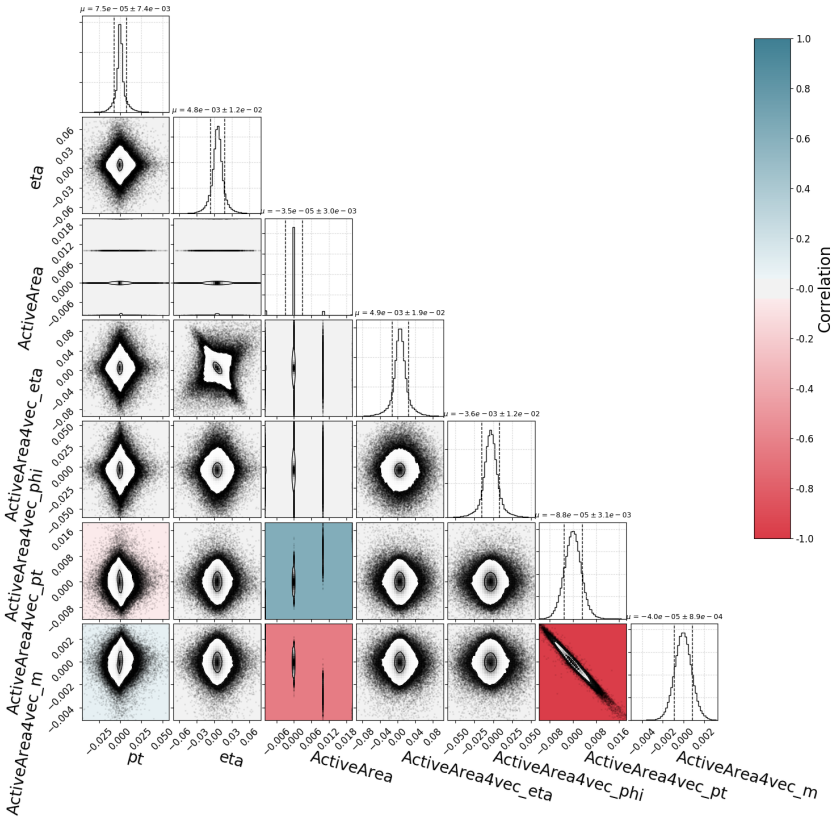

Figure A.9: Residuals of the 18D latent space AE. Values shown are relative difference for m and p_T, absolute difference for η and ϕ. Axes have been scaled to include 99.9% of all jets.

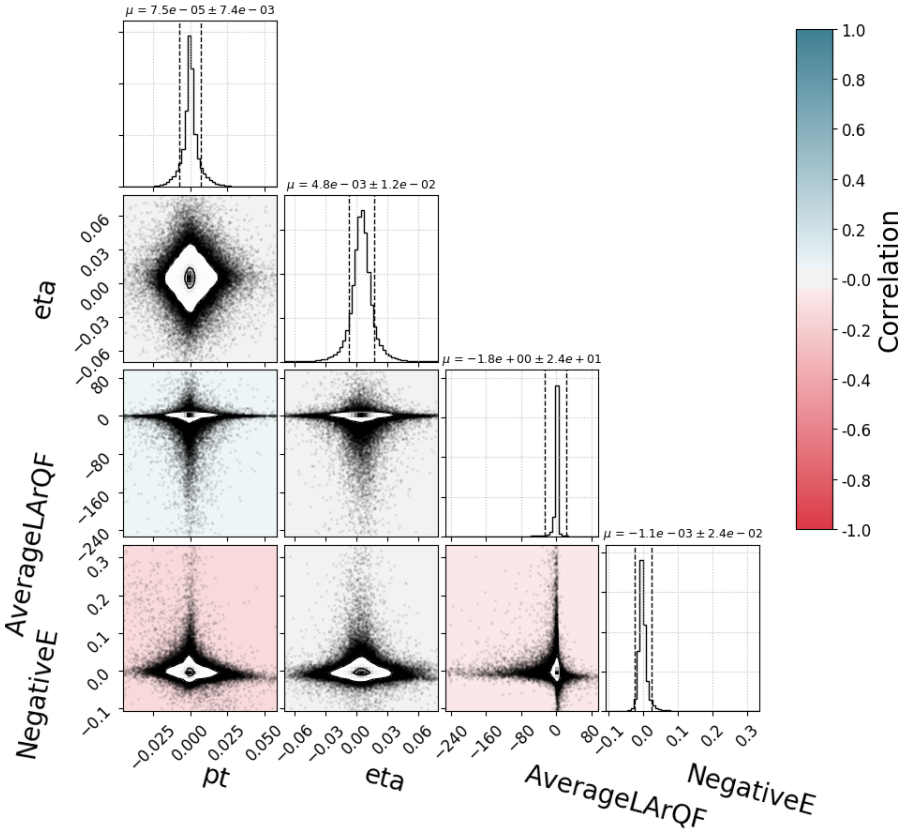

Figure A.10: Residuals of the 18D latent space AE. Values shown are relative difference for m and p_T, absolute difference for η and ϕ. Axes have been scaled to include 99.9% of all jets.

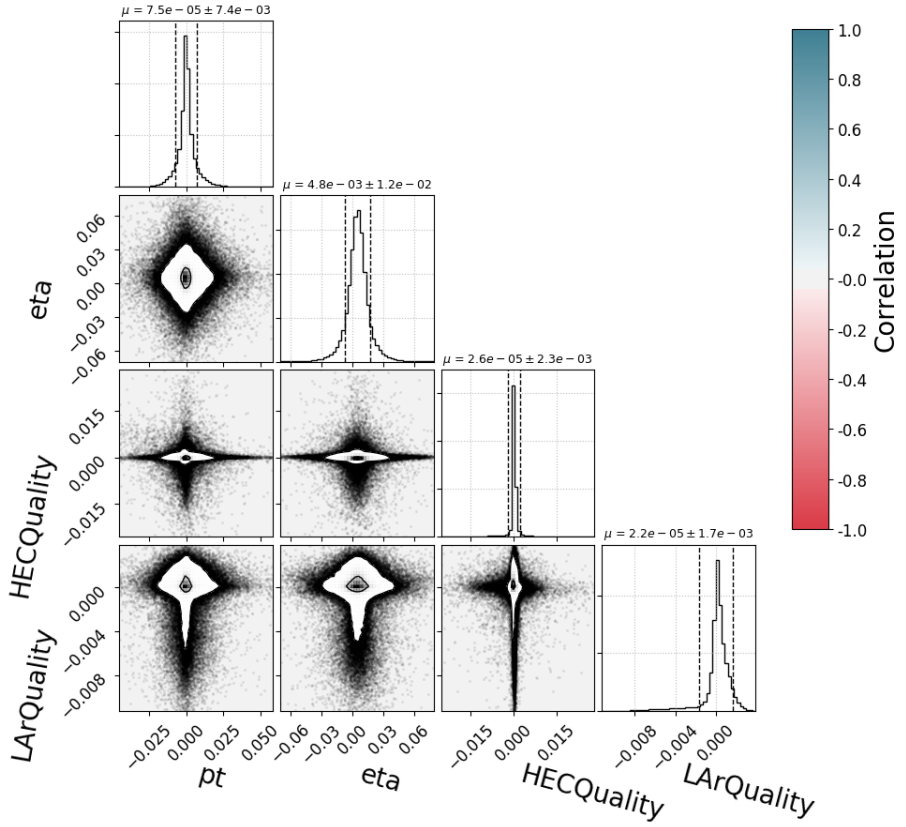

Figure A.11: Residuals of the 18D latent space AE. Values shown are relative difference for m and p_T, absolute difference for η and ϕ. Axes have been scaled to include 99.9% of all jets.

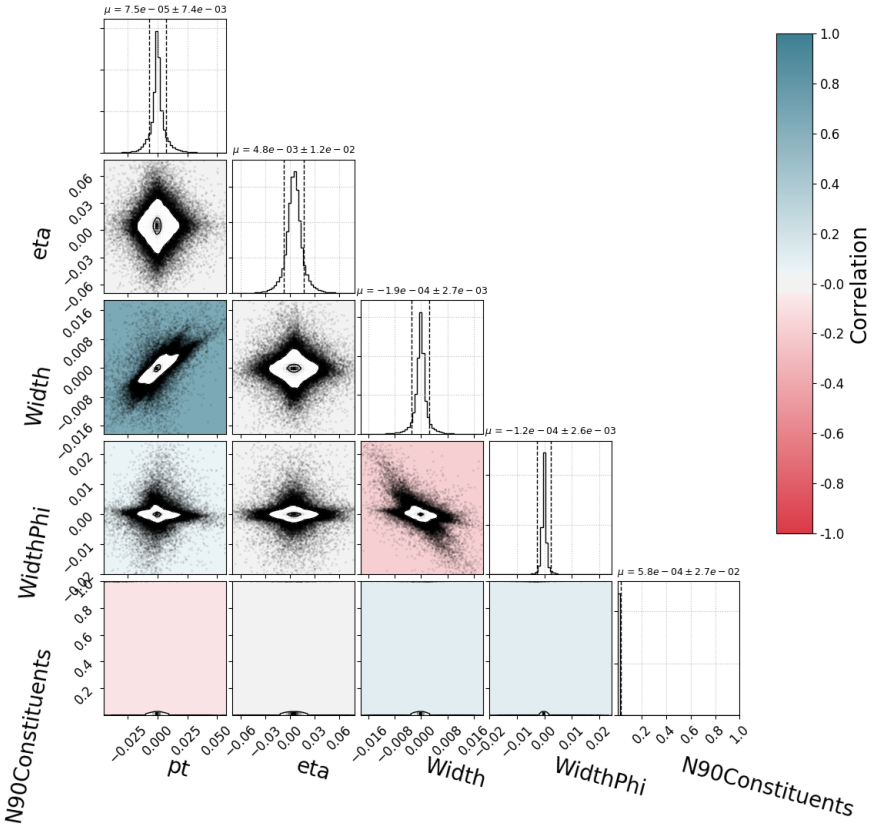

Figure A.12: Residuals of the 18D latent space AE. Values shown are relative difference for m and p_T, absolute difference for η and ϕ. Axes have been scaled to include 99.9% of all jets.

A.5 Normalized input for the 27 variable input AEs

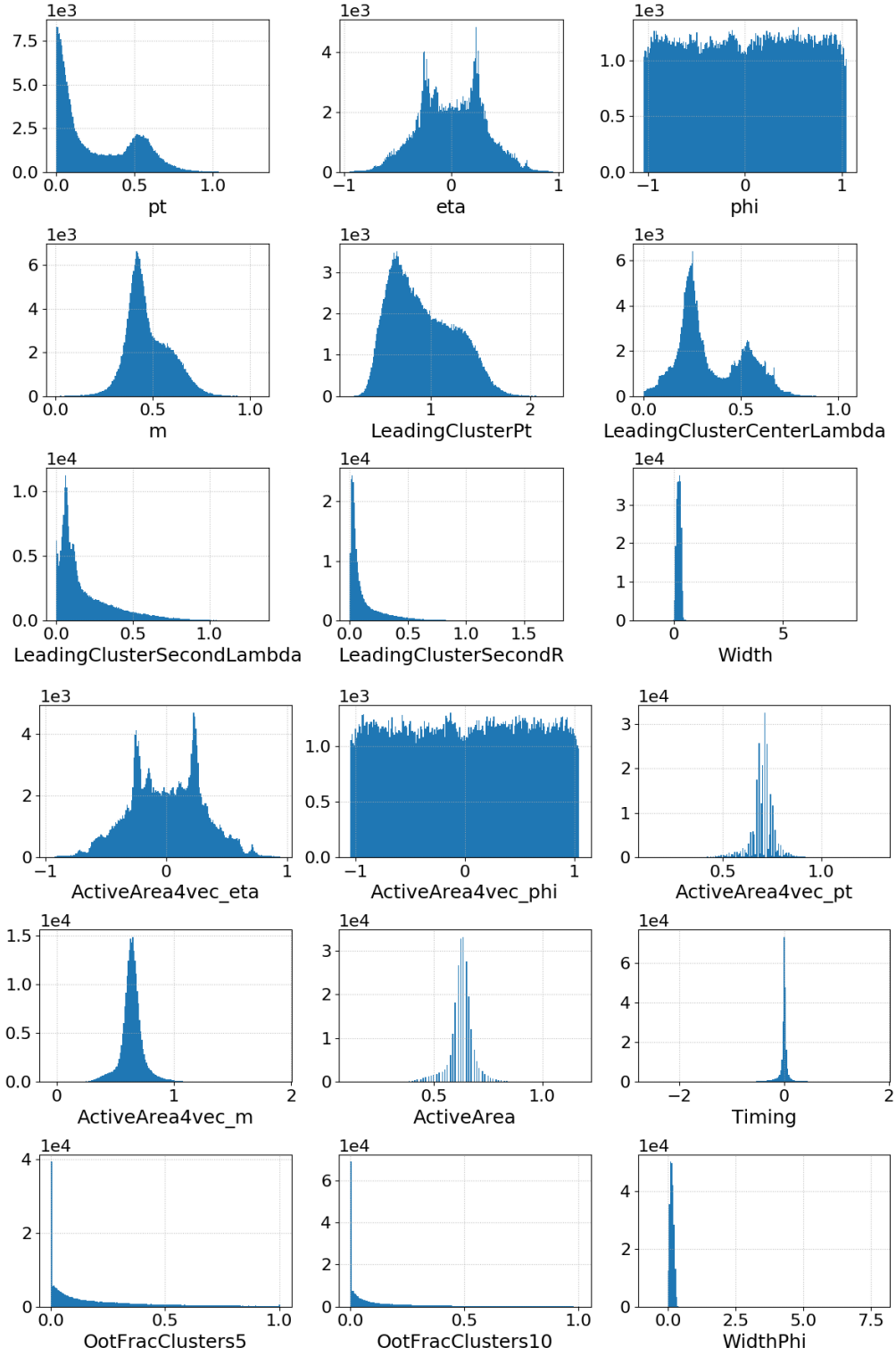

Figure A.13: Distributions of a subset of the 27 variables after the custom normalization.

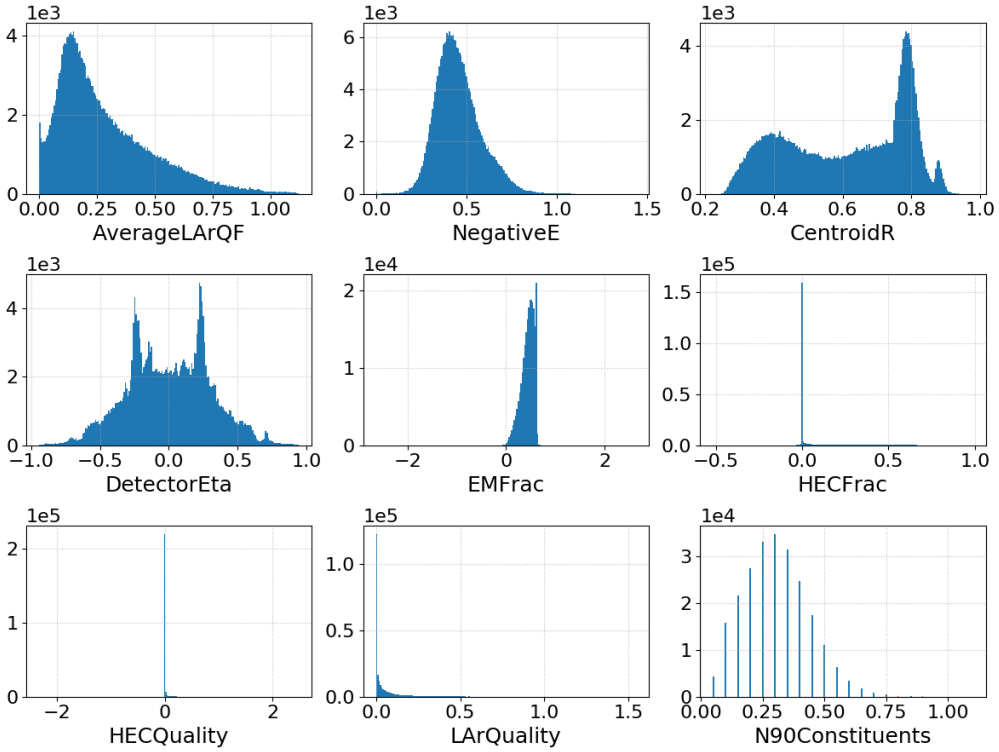

Figure A.14: Distributions of a subset of the 27 variables after the custom normalization.