Deep autoencoders for ATLAS data compression

George Dialektakis

Google Summer of Code 2021

Mentors: Alex Gekow, Caterina Doglioni, Baptiste Ravina, Antonio Boveia

Jun-Aug, 2021

Contents

Chapter 1

Abstract

Storage is one of the main limiting factors to the recording of information from proton-proton collision events at the Large Hadron Collider (LHC), at CERN in Geneva. Hence, the ATLAS experiment at the LHC uses a so-called trigger system, which selects and transfers interesting events to the data storage system while filtering out the rest. However, if interesting events are buried in very large backgrounds and difficult to identify as a signal by the trigger system, they will also be discarded together with the background. To alleviate this problem, different compression algorithms are already in use to reduce the size of the data that is recorded. One of those state-of-the-art algorithms is an autoencoder network that tries to implement an approximation to the identity, $f(x) = x$, and given some input data, its goal is to create a lower-dimensional representation of those data in a latent space using an encoder network. This way when collisions happen on the ATLAS Collider, we run the encoder on the produced data and we save only the latent space representation. Then using this latent representation offline the decoder network can reconstruct the original data.

The goal of this project is to experiment with different types of Autoencoders for data compression in-depth and optimize their performance in reconstructing the ATLAS event data. For this reason, three kinds of Autoencoders are proposed, and in specific, the Standard Autoencoder, the Variational Autoencoder, and the Sparse Autoencoder. The above Autoencoders and thoroughly tested using different parameters and data normalization techniques, as our ultimate goal is to obtain the best possible reconstructions of the original event data. The proposed implementations will be a decisive contribution towards future testing and analysis for the ATLAS experiment at CERN and will assist overcome the obstacle of needing much more storage space than in the past due to the increase in the size of the data generated by the continuous proton-proton collision events in CERN's Large Hadron Collider.

Chapter 2

Project Description

At CERN's Large Hadron Collider (LHC) [1], proton collisions are performed to study the fundamental particles and their interactions. To discover and record the outcome of these collisions, the ATLAS detector [2] is used, which is one of many detectors that have been developed at the LHC. Each second, there are roughly 10^9 collisions occurring inside the ATLAS detector and storage is one of the main limiting factors to the recording of information from these events, since it is impossible to save all those events. To keep the most relevant information, the ATLAS experiment uses trigger systems, which selects and transfers interesting events to the data storage system while filtering out the rest. Storage of these events is restricted by the amount of information to be stored and a decrease of the event size can allow for an analysis of the events that was not previously possible.

Data compression is a solution to the above problem by encoding information using fewer dimensions or a smaller size than the original representation [3]. One of the current state-of-the-art data compression techniques is deep compression using Autoencoders [4]. Typically, an Autoencoder (AE) is an Artificial Neural Network (ANN) that tries to implement an approximation of the identity function. It consists of an encoder that performs data compression by projecting the input high dimensional data to a lower-dimensional latent space, and a decoder that performs the reversal of the process (decompression) by projecting the latent space back to the dimensions of the original data, also known as the reconstruction phase. The latent space representation can then be used as a compressed representation of the input and can be stored along with the decoder network to reconstruct the data.

In this project, we examine and experiment on different types of Autoencoders as we try to effectively compress the original data into lower dimensional spaces and learn meaningful features.

The structure of this proposal is organized as follows. Section 3 presents related work in the field of HEP data compression using Autoencoders. In section 4, we discuss the data that we use for this analysis along with the preprocessing steps we follow. In section 5, we explain the different types of Autoencoders that we use for this project. Section 6 presents the experimental process and the results we obtain. Finally, the last section of this report discusses some conclusions that are drawn and possible ideas for the future.

Chapter 3

Related Work

Previous work [5] examines the use of deep neural autoencoders to compress data for jets. Jets are collimated streams of particles, mostly hadrons i.e. bound states of quarks, that can be reconstructed as a single object by clustering all these particles together. Different autoencoder variants are tested with different widths and depths. The work shows promising results as the AEs manage to successfully compress and decompress simple jet data and preliminary results indicate that the reconstruction quality is good enough for certain applications where high precision is not paramount. The work in [6] being the most relevant reference to this work, further investigates the use of AEs for compression of trigger level analysis (TLA) data, which come from an on-the-fly, rough analysis of the data, as compared to the offline analysis performed with carefully reconstructed and calibrated particles, as used by the ATLAS experiment, while showing that it may however be difficult to generalize between different datasets. Moreover, the use of different compression methods used sequentially, by so-called float truncation then followed by autoencoder compression is evaluated.

Chapter 4

Data & Preprocessing

The data for this project originate from a file named 00992A80-DF70-E211-9872-0026189437FE.root (http://opendata.cern.ch/record/6010) from JetHT primary dataset in AOD format from RunB of 2012 from the Compact Muon Solenoid (CMS) Experiment at CERN [7]. This dataset is a public dataset from the CMS experiments composed of mostly of jets, each one described by 28 variables. The distribution of those 28 variables of our data can be seen in fig. 4.2, 4.3.

Before feeding our data in any machine learning algorithm, we first need to preprocess them. In specific, we first filter out those jets that have pT $>$ 8 TeV and mass $<$ 800 GeV, as they are considered outliers. While this is not the full set of preprocessing that the CMS collaboration performs to obtain a dataset ready for physics analysis, it is sufficient for our purposes to remove events that could have originated from noise such as those ones with extremely high energies. Then we remove variables fX, fY, fZ, and mPileupEnergy as they consist of only zero values and therefore we do not need to compress them, leading to 24 variables in our data. Next, we consider two variations of the data, the 24D and the 19D data. The 24D are the initial variables after we remove the four above variables. The 19D data are composed of the same variables as the 24D data except for mChargedEmEnergy, mChargedMuEnergy, mMuonEnergy, mMuonMultiplicity, and mElectronEnergy, which are removed as the 99.5% of them in the data have a zero value.

The last step of our preprocessing procedure is normalizing the data. For this task, we have considered three different techniques so as to evaluate which is the best suited for our problem. The first one is standard scaling which standardizes features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

$$z = (x - u)/s \tag{4.1}$$

where u is the mean of the training samples, and s is the standard deviation of the training samples. If the input distribution is Gaussian, Standard Scaling leads to a Normal distribution (mean=0, std=1) which is more suited for Machine Learning. However, as can be seen from the plots in fig. 4.2, 4.3, most of the variables are not quite Gaussian, so it may not be so effective.

The second normalization technique we considered is MinMax Scaling which transforms features by scaling each feature to a given range as shown in eq. 4.2. This estimator scales and translates each feature individually such that it is in the given range on the training set, in our case between zero and one.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{4.2}$$

The last normalization method we used is a combination of MinMax Scaling and a custom normalization for the 4-momentum of our data (i.e. pT, phi, eta, and mass). In specific, we pursued the following equations for the 4-momentum variables, while the rest of the variables are scaled between the ranged of [0, 1]:

$$\begin{aligned} p_T &\longrightarrow \log_{10}\left(p_T - 1.3\right)/1.8 \\ E &\longrightarrow \log_{10}(E+1)/1.2 \\ \eta &\longrightarrow \eta/5 \\ \phi &\longrightarrow \phi/3 \end{aligned} \tag{4.3}$$

In figure 4.1 we observe the correlation between the 28 initial variables in our data.

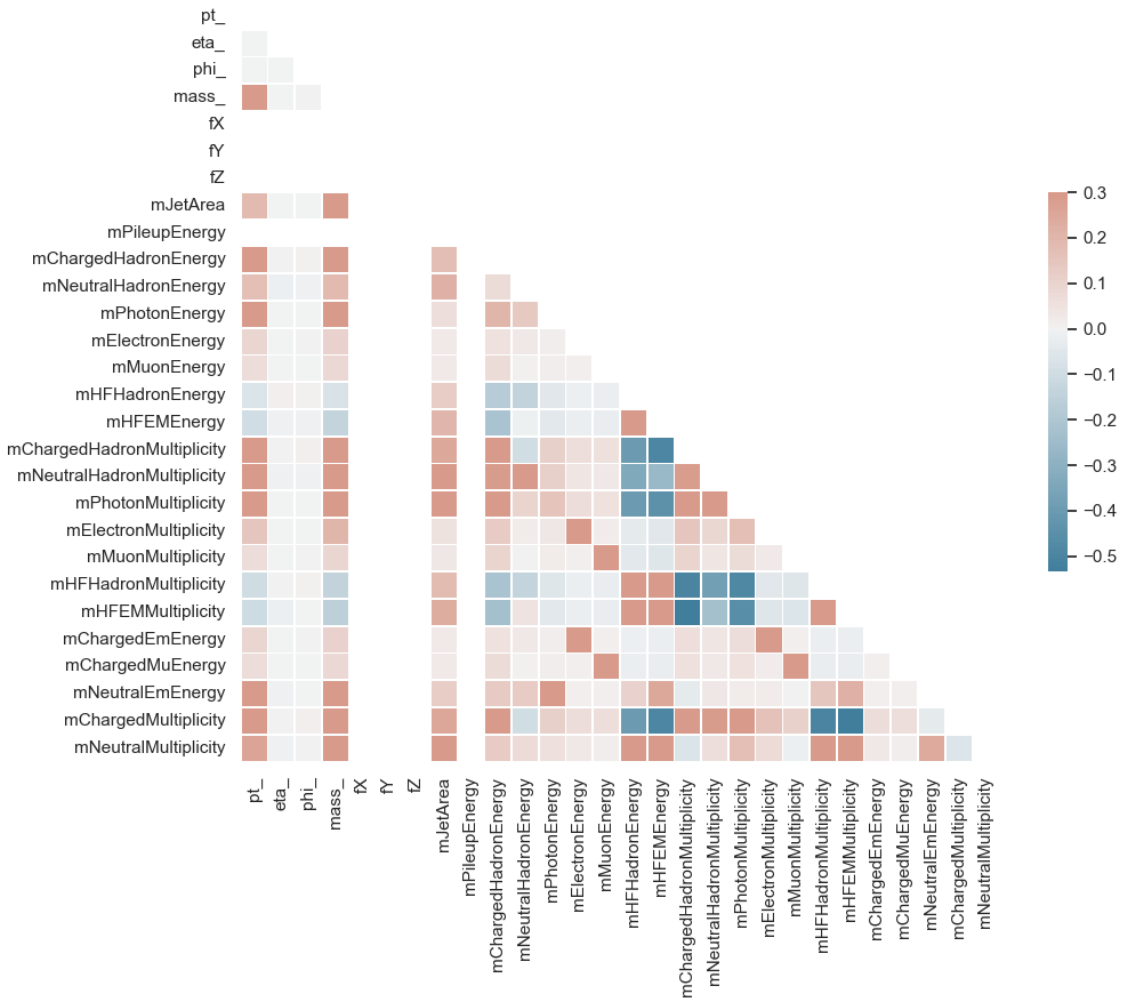
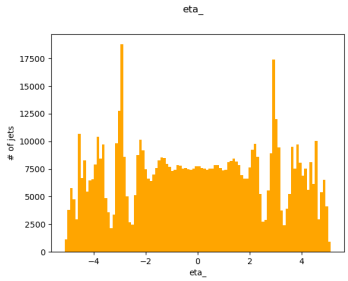
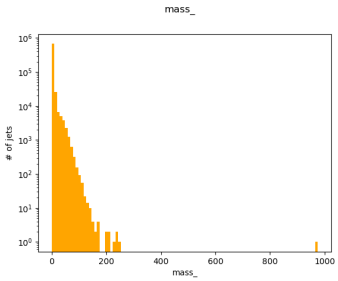


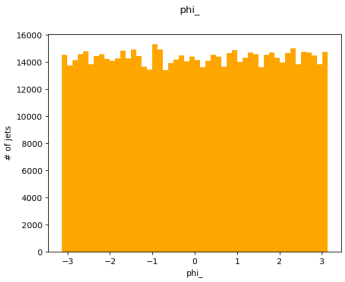
Figure 4.1: Correlation between the 28 variables of the initial raw data.



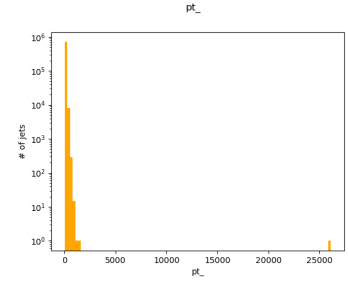
(a) eta



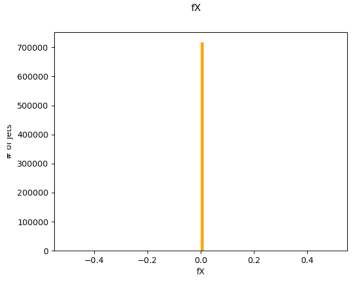
(b) mass



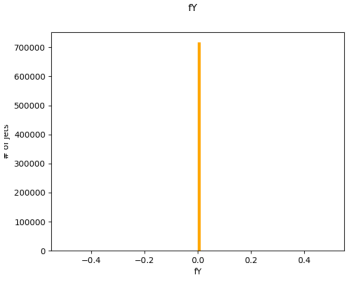
(c) phi



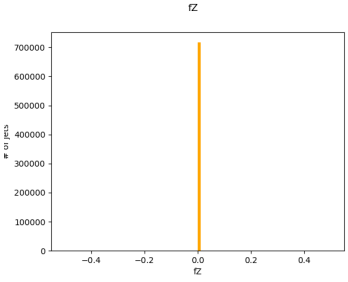
(d) pt



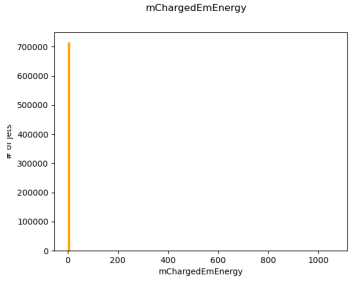
(e) fX



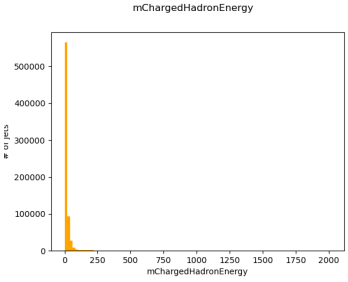
(f) fY



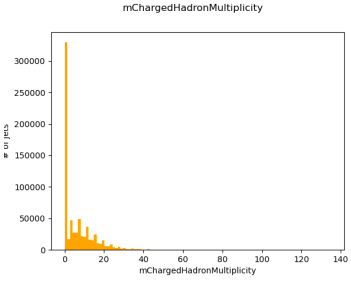
(g) fZ



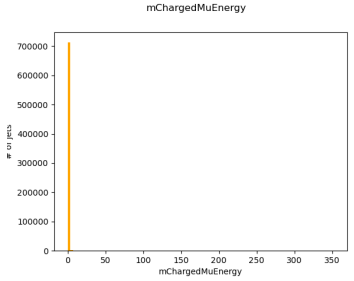
(h) mChargedEmEnergy



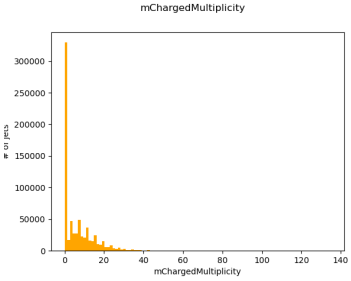
(i) mChargedHadronEnergy



(j) mChargedHadronMultiplicity

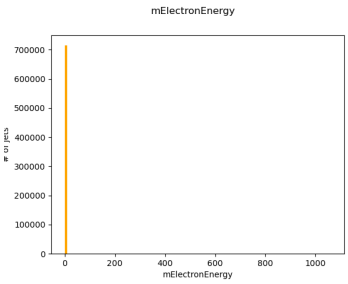


(k) mChargedMuEnergy

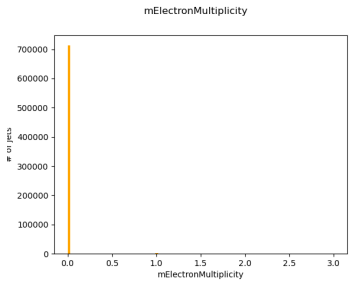


(l) mChargedMultiplicity

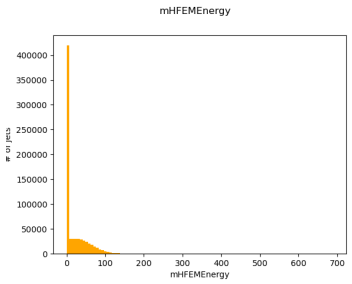
Figure 4.2: Distributions of the 28 initial variables



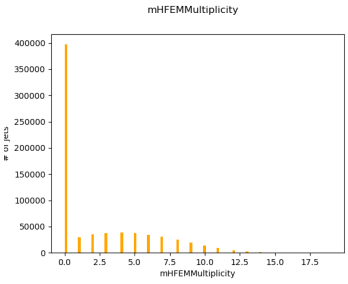
(a) mElectronEnergy



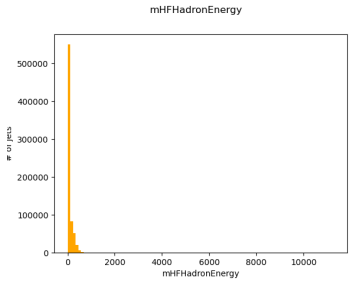
(b) mElectronMultiplicity



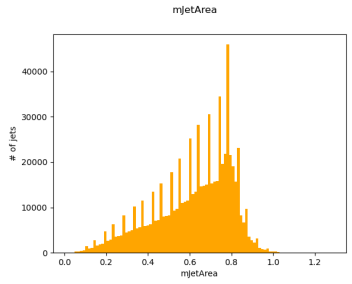
(c) mHFEMEnergy



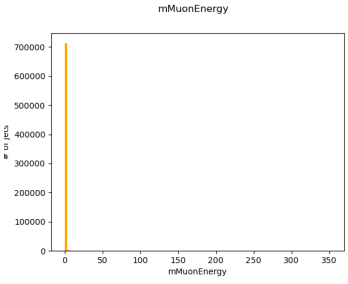
(d) mHFEMMultiplicity



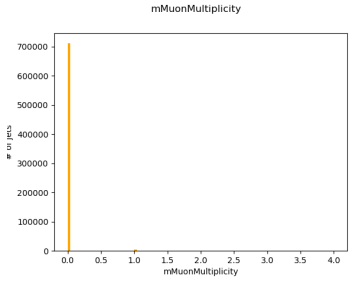
(e) mHFHadronEnergy



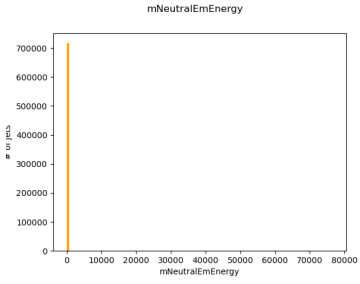
(f) mJetArea



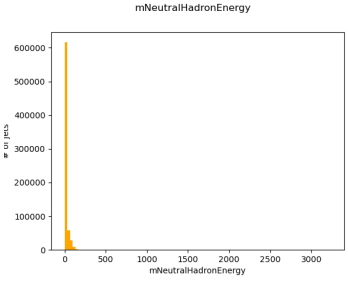
(g) mMuonEnergy



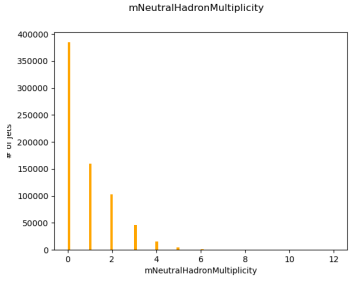
(h) mMuonMultiplicity



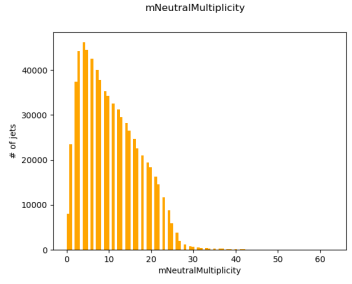
(i) mNeutralEmEnergy



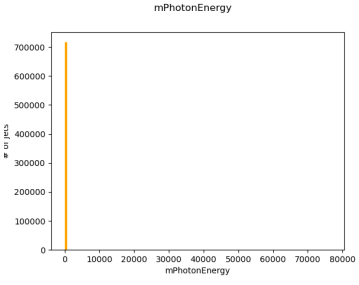
(j) mNeutralHadronEnergy



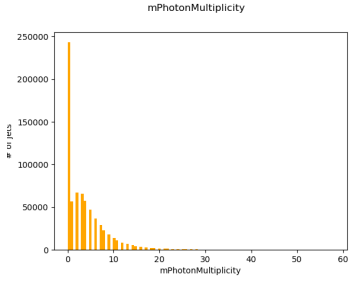
(k) mNeutralHadronMultiplicity



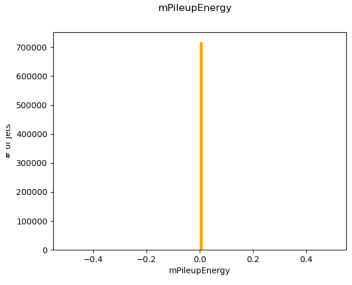
(l) mNeutralMultiplicity



(m) mPhotonEnergy



(n) mPhotonMultiplicity



(o) mPileupEnergy

Figure 4.3: Distributions of the 28 initial variables

Chapter 5

Autoencoders

5.1 Standard Autoencoder

Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of representation learning [8]. Specifically, an Autoencoder is a neural network architecture with a bottleneck layer in the middle of the network which forces a compressed knowledge representation of the original input. If the input features were each independent of one another, this compression and subsequent reconstruction would be a very difficult task. However, in this work we deal with a physics dataset, where we know that structure exists in the input data (i.e. correlations between input features). This structure can be learned and consequently leveraged when forcing the input through the network's bottleneck.

A simple Autoencoder consists of three components: an Encoder network, a Decoder network, and a bottleneck layer called latent code, as shown in fig. 5.1

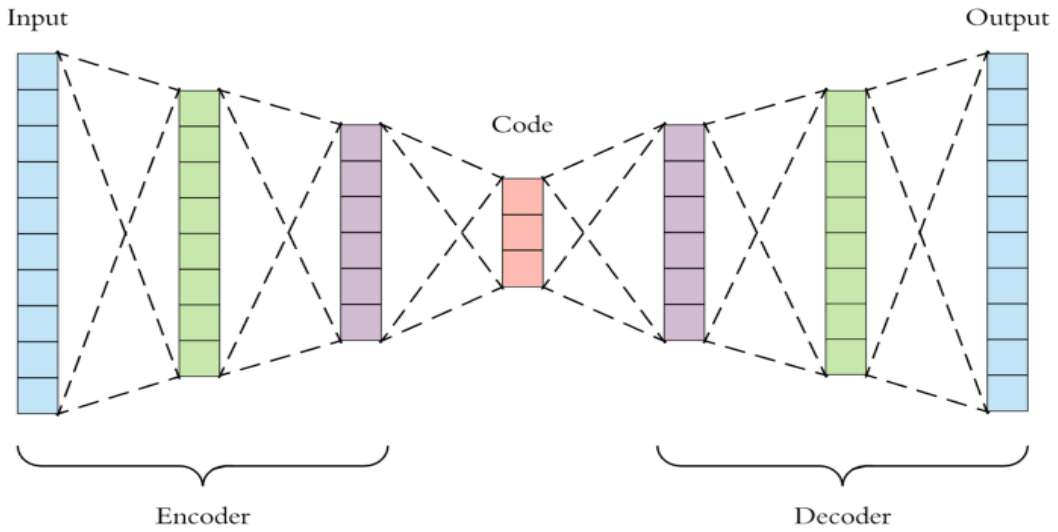


Figure 5.1: Standard Autoencoder architecture [9]

First the input passes through the encoder, which is a fully-connected ANN, to produce the code. The decoder, which has the same ANN structure with the Encoder but mirrored, then produces the output only using the code. The goal is to get an output identical with the input. To achieve such functionality, the Autoencoder uses the Mean Squared Error (MSE) as a loss function which measures the distance between the input data and the predicted (reconstructed) data. In that way, the network is penalized when producing predictions that are far from the input data and forces the Autoencoder to learn better and more robust lower-dimensional representations of the input.

5.2 Variational Autoencoder

Variational Autoencoder (VAE) forms a generative model composed of an encoder and a decoder trained to minimize the reconstruction error between the output of the decoder and the input of the encoder. However, it differs from standard autoencoders, as it performs a regularization technique in the latent encoding. Specifically, every input of the VAE is encoded as a distribution over the latent space rather than a single point. Next, a point from the latent space is sampled from that distribution, which is then fed to the decoder to compute the reconstruction error and backpropagate it through the network. The architecture of the VAE is illustrated in Figure 5.2.

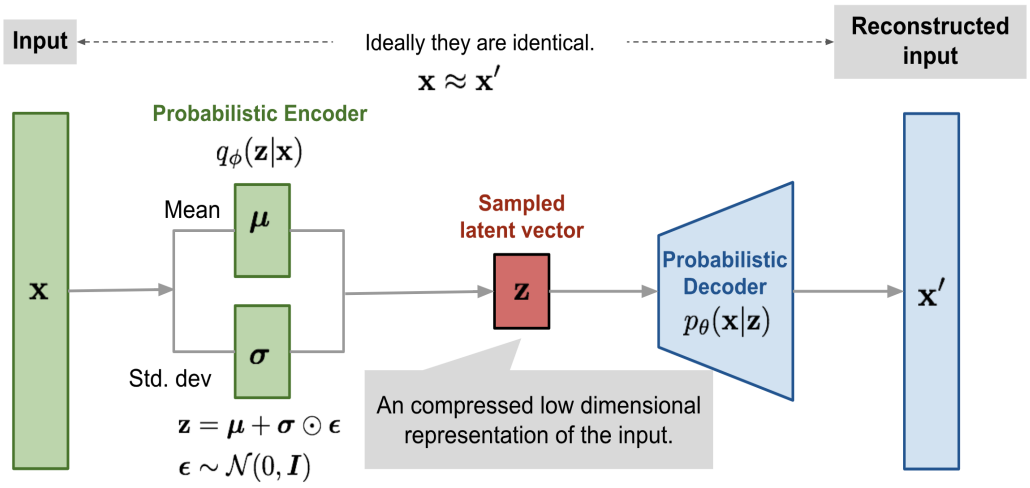


Figure 5.2: Variational Autoencoder architecture [10]

The encoded distributions are chosen to be normal, described by a vector of means $\boldsymbol{\mu}$ and a vector of standard deviations $\boldsymbol{\sigma}$. The vector of means controls the center around which an input is encoded, and the standard deviation controls the space, how far from the mean the encoding can be. VAE ideally wants to produce encodings that are as close as possible to each other while still remaining separated. In this way, it allows smooth interference and enables

the generation of new samples. To achieve the regularization functionality, VAEs incorporate the Kullback–Leibler [11] divergence into their loss function. The KL divergence is applied between the distribution produced by the encoder and a standard Gaussian, in order to force the encoder to create encodings that follow a normal distribution. The complete cost function of VAEs is shown below:

$$\begin{aligned} E_{x\sim p_d(x)}[-log\ p(x)] &< E_x[E_{q(z|x)}[-log(p(x|z)]] + E_x[\boldsymbol{KL}(q(z|x)||p(z))] \\ &= E_x[E_{q(z|x)}[-log(p(x|z)]] - E_x[H(q(z|x))] + E_{q(z)}[-log\ p(z)] \\ = E_x[E_{q(z|x)}[-log(p(x|z)]] - E_x[\sum_i log\sigma_i(x)] &+ E_q(z)[-log\ p(z)] + const. \\ &= Reconstruction - Entropy + CrossEntropy(q(z), p(z)) \end{aligned} \tag{5.1}$$

where the aggregated posterior $q(z)$ is sampled from the output of the encoder $q(z|x)$, and $p(z)$ is usually a standard Gaussian distribution.

5.3 Sparse Autoencoder

Sparse Autoencoder is a special type of Autoencoders which is able to maintain only the variations in the data required to reconstruct the input without holding on to redundancies within the input. This way the Sparse Autoencoder (SAE) can be sensitive enough to the inputs to accurately build a reconstruction, and at the same time insensitive enough to the inputs so that it doesn't simply memorize or overfit the training data [12]. To achieve such functionality, SAE uses a loss function composed of two terms as shown in eq. 5.2

$$\mathcal{L}(x, \hat{x}) + \text{ regularizer} \tag{5.2}$$

The first term is the reconstruction loss L(x,x̂) which encourages the model to be sensitive to the inputs, while the second term discourages memorization/overfitting (i.e. an added regularizer). The loss function of a Sparse Autoencoder is constructed so that it penalizes activations within a layer. For any given observation, the network is encouraged to learn an encoding and decoding which only relies on activating a small number of neurons. It's worth noting that this is a different approach towards regularization, as we normally regularize the weights of a network, not the activations. One result of this fact is that we allow our network to sensitize individual hidden layer nodes toward specific attributes of the input data by selectively activating regions of the network depending on the input data. As a result, we've limited the autoencoder's capacity to memorize the input data without limiting the its capability to extract features from the data. This allows us to consider the latent state representation and regularization of the network separately, such that we can choose a latent state representation (i.e. encoding dimensionality) in accordance with what makes sense given the context of the data while imposing regularization by the sparsity constraint.

There are two main ways by which we can impose this sparsity constraint; both involve measuring the hidden layer activations for each training batch and adding some term (the regularizer in eq. 5.2) to the loss function in order to penalize excessive activations. These terms are:

- **L1 Regularization:** We can add a term to our loss function that penalizes the absolute value of the vector of activations a in layer h for observation i, scaled by a tuning parameter λ.

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i \left| a_i^{(h)} \right| \tag{5.3}$$

- **KL-Divergence:** In essence, KL-divergence is a measure of the difference between two probability distributions. We can define a sparsity parameter ρ which denotes the average activation of a neuron over a collection of samples. This expectation can be calculated as $\hat{\rho}_j = \frac{1}{m} \sum_i \left[a_i^{(h)}(x) \right]$ where the subscript j denotes the specific neuron in layer h, summing the activations for m training observations denoted individually as x. In essence, by constraining the average activation of a neuron over a collection of samples we're encouraging neurons to only fire for a subset of the observations. We can describe ρ as a Bernoulli random variable distribution such that we can leverage the KL divergence to compare the ideal distribution ρ to the observed distributions over all hidden layer nodes $\hat{\rho}$.

$$\mathcal{L}(x, \hat{x}) + \sum_j KL\left(\rho \| \hat{\rho}_j\right) \tag{5.4}$$

Chapter 6

Experiments

6.1 Setup & Parameter Tuning

For our experimental analysis, we constructed all our Autoencoders to have the same architecture in terms of layers and neurons per layer. In specific all networks share the following architecture:

- input-200-100-50-**15**-50-100-200-output

where the output dimension is the same as the input dimension and it is either 19D or 24D depending on the variables we want to compress. All layers in the Autoencoders are fully connected and are followed by a LeakyRelu activation function. We used the Adam optimizer with a learning rate of 0.001 and trained our networks for 50 epochs. The data are composed of 716445 samples and they were split with a 15% ratio (85% for training and 15% for testing) and a batch size of 512 was used.

6.2 Autoencoders Results

6.2.1 Data Normalization Comparison

In the first series of experiments we wanted to test how the different data normalization methods affects the performance of our Autoencoders in compressing the data and producing valuable reconstructions of the original input data. For this reason, we use the three different techniques as described in Chapter 4, to normalize our data both the 19D and 24D and then feed those normalized data into the Standard Autoencoder. In figure 6.1 we observe the Mean Squared Error loss (MSE) of the Standard Autoencoder on the test data. It is shown that Standard AE performs the worst (as it has the greatest MSE loss) when the normalized data with the Standard Scaler is given as input, while it performs the best in the case of the MinMax Scaling of the data in the range [0, 1]. This performance behaviour is consistent in both cases, when the data consists of 19 variables and 24 variables. Apart from a quite low MSE loss, MinMax scaled data provide the Standard AE a much smoother learning

process as illustrated in fig. 6.2. It is also proved that applying a custom normalization on 4-momentum variables did not help the Autoencoder.

A more detailed performance comparison of the normalization methods considered is present in table 6.1, where apart from MSE loss, two more metrics are used to evaluate the performance of the Standard Autoencoder, the Root Mean Squared Error (RMSE) and the Residuals which is the difference between the input and the output of the network. As scaling all the variables in our data in the range [0, 1] shows to help the Standard AE achieve the minimum loss, we decide this method for the rest of the Autoencoders we consider in this work so as to unlock their the full potential.

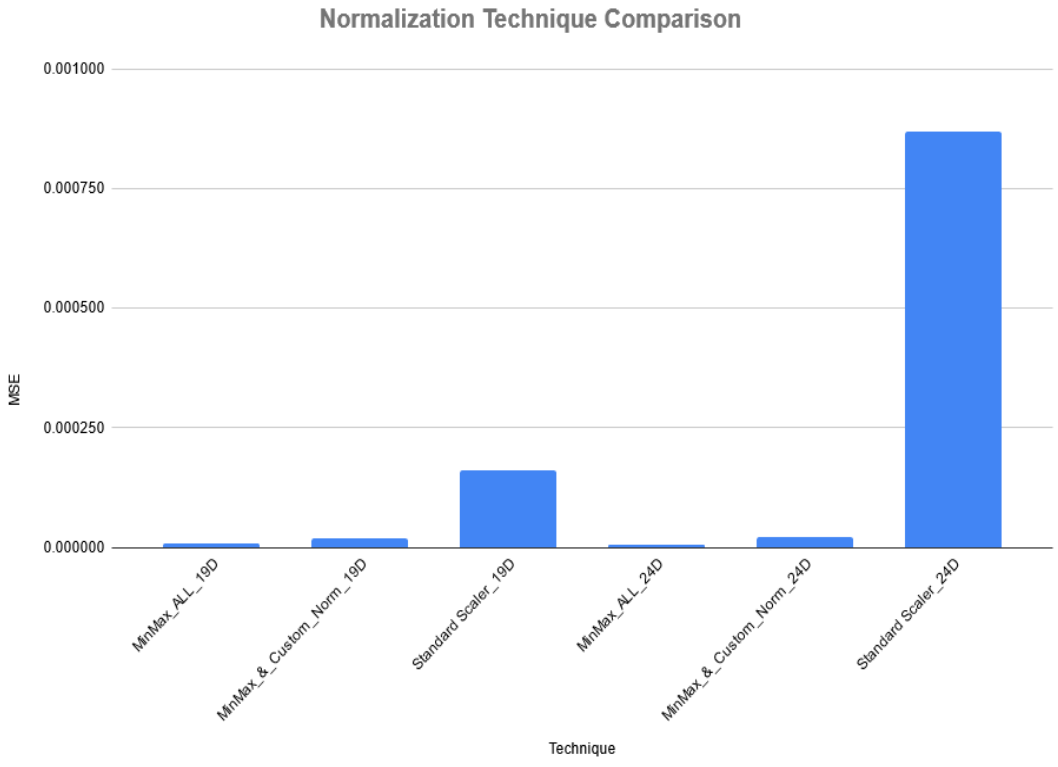
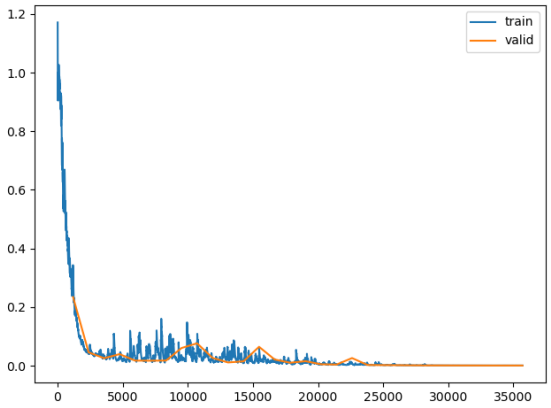
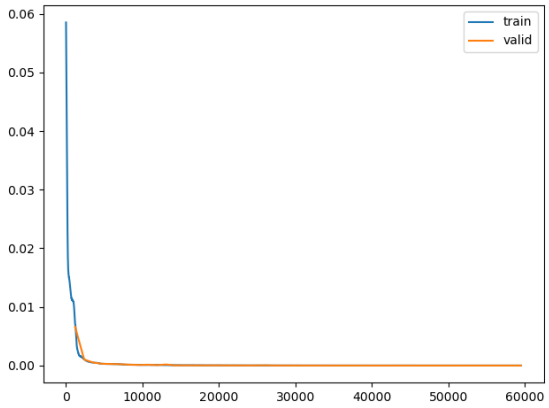


Figure 6.1: Data Normalization Technique Comparison.



(a) Standard Scaler



(b) MinMax Scaler

Figure 6.2: Learning curves of Standard AE using Standard Scaled vs MinMax Scaled data.

Dimension	Technique	MSE	RMSE	Residuals
19	**MinMax_ALL_19D**	**0.000008**	**0.002677**	**0.001249**
	MinMax_&_Custom_Norm_19D	0.000019	0.003854	0.002184
	Standard Scaler_19D	0.000162	0.012496	0.007408
24	**MinMax_ALL_24D**	**0.000007**	**0.002460**	**0.001040**
	MinMax_&_Custom_Norm_24D	0.000023	0.004306	0.002091
	Standard Scaler_24D	0.000870	0.028226	0.011937

Table 6.1: Data Normalization Technique Comparison.

6.2.2 Sparse Autoencoder Results

In figure 6.3 we present the MSE loss comparison between the two variants of the Sparse Autoencoder, the one that uses KL divergence (SAE_KL) and the other that uses the L1 regularization (SAE_L1) for the 19D and 24D data, as described in 5.3. The SAE with L1 regularization performs much better than the one that utilizes KL divergence in both data dimensions we considered. In specific, the MSE loss of the SAE with L1 is roughly 35 times smaller than that of the the SAE with KL divergence. A summary of the performance results for the two Sparse Autoencoders are presented in Table 6.2, which confirms the better performance of the SAE with L1 regularization.

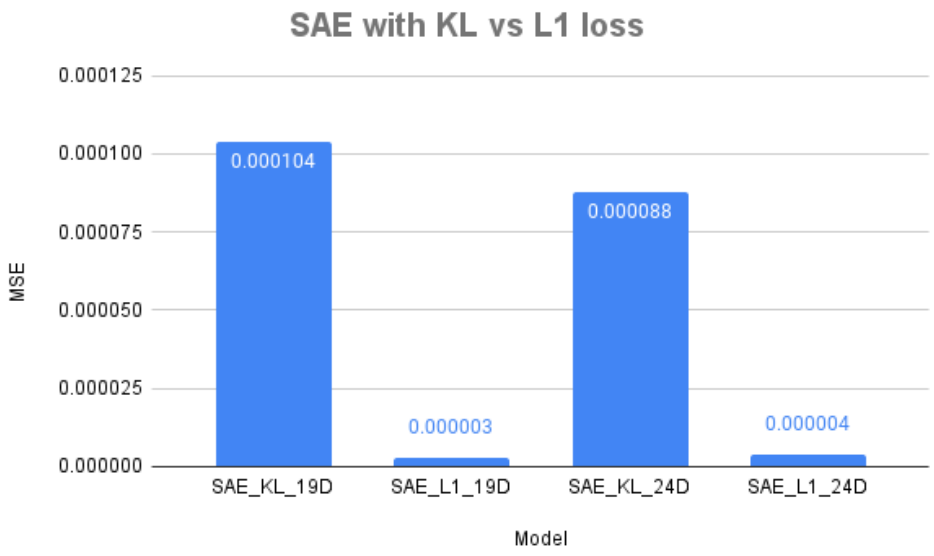


Figure 6.3: MSE loss comparison for Sparse Autoencoder

Dimension	Model	MSE	RMSE	Residuals
19	SAE_KL_19D	0.000104	0.009332	0.004724
	SAE_L1_19D	**0.000003**	**0.001630**	**0.000933**
24	SAE_KL_24D	0.000088	0.008493	0.003868
	SAE_L1_24D	**0.000004**	**0.001778**	**0.001247**

Table 6.2: Sparse Autoencoder using KL vs L1 regularization loss for 19D and 24D data. Bold shows the best results.

6.2.3 Variational Autoencoder Results

In figure 6.4 we illustrate the performance results of the Variational Autoencoder in terms of MSE, RMSE loss and Residuals. VAE shows to perform a little better when it is given the 24D data. However, the loss of the VAE is quite high compared to the previous results produced by the Standard AE and the Sparse AE, showing a weakness of the VAE to learn meaningful patterns of the input variables and effectively compress them in a lower dimensional space. This results in poor reconstructions as depicted in fig. 6.5 (for the sake of brevity we present only the plots of the 4-momentum, however the same performance behaviour stands for the rest of the variables.)

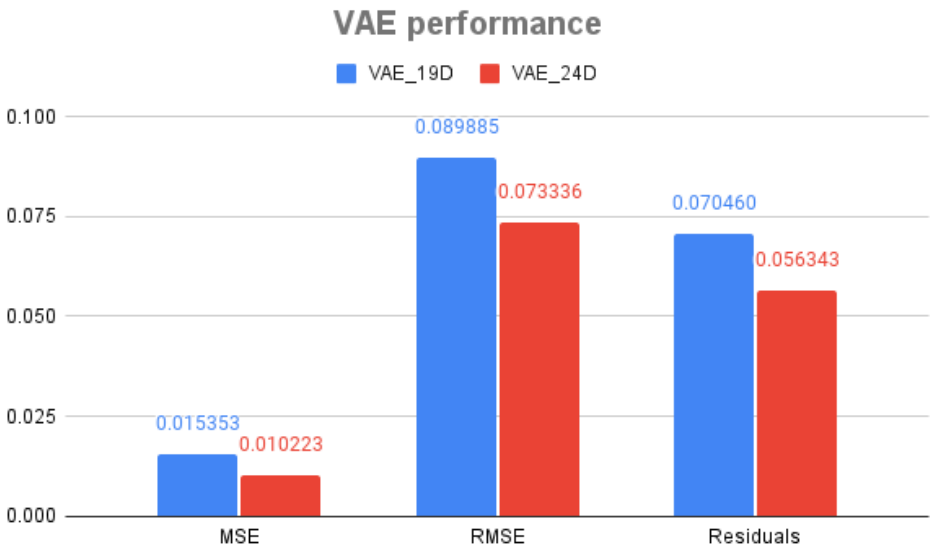
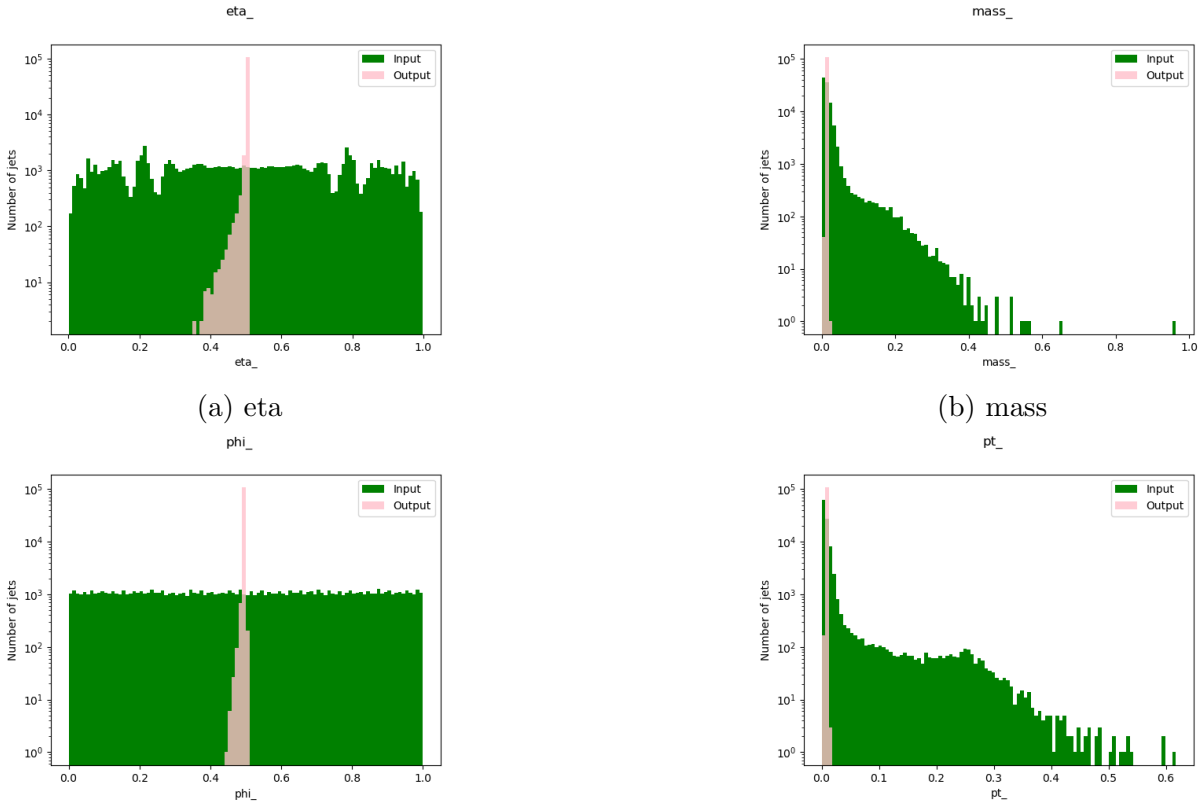


Figure 6.4: Performance of Variational Autoencoder for 19D and 24D data.



(a) eta

(b) mass

(c) phi

(d) pT

Figure 6.5: VAE reconstructions of the 4-momentum variables.

6.2.4 Autoencoders Performance Comparison

In this section we summarize the overall performance results of the three different Autoencoder families we implemented and tested throughout this project. In Table 6.3 the MSE, RMSE, and Residuals are presented for the three different Autoencoders, the Standard Autoencoder (AE), the Sparse Autoencoder (SAE), and the Variational Autoencoder (VAE) for the 19D and 24D data. For the SAE we choose the network that utilizes the L1 regularization as it proved to perform much better than the one with the KL divergence as discussed in 6.2.2. It is apparent that VAE shows quite poor performance compared to the other two AEs. A reason for that could be the KL divergence term in its loss function which also showed to degrade the performance in the case of the SAE.

Dimension	Model	MSE	RMSE	Residuals
19	**SAE_19D**	**0.000003**	**0.001630**	**0.000933**
	AE_19D	0.000008	0.002677	0.001249
	VAE_19D	0.015353	0.089885	0.070460
24	**SAE_24D**	**0.000004**	**0.001778**	**0.001247**
	AE_24D	0.000007	0.002460	0.001040
	VAE_24D	0.010223	0.073336	0.056343

Table 6.3: Performance comparison between all different Autoencoders considered. Best result in bold.

Since the AE and the SAE have close performance results, we compare their MSE loss in the bar plot in fig. 6.6. We observe that the SAE leads to lower MSE loss in both 19D and 24D data. Specifically, SAE's loss is almost the half of the AE's loss, proving that adding the sparsity constraint in the Autoencoder helps the network to train better and learn to compress the input data more effectively. Finally, in figures 6.7, 6.8 we present a comparison between the reconstructions of the the Sparse and the Standard AE.

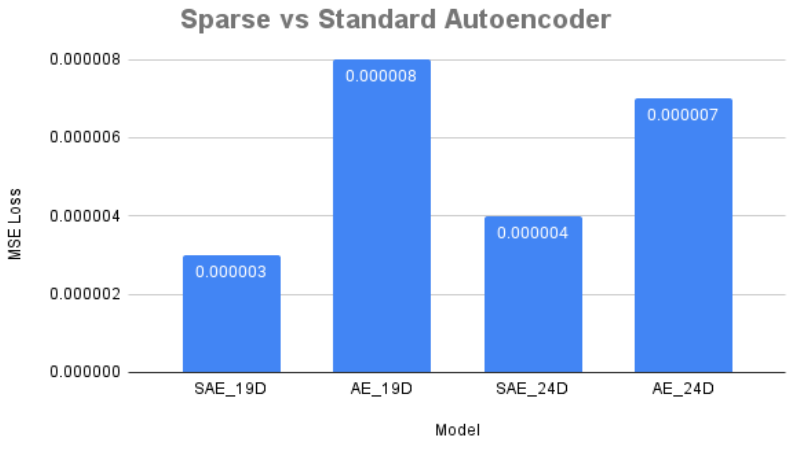
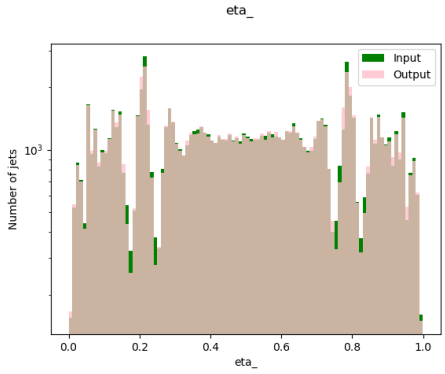
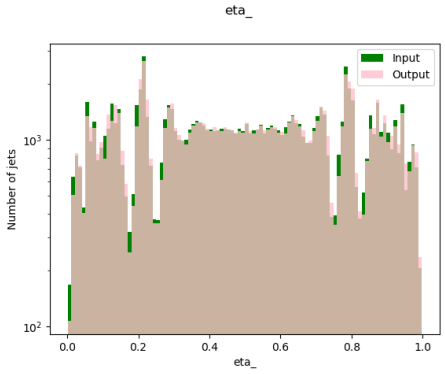


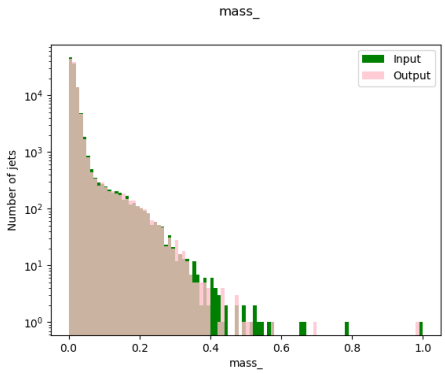
Figure 6.6: Sparse vs Standard Autoencoder MSE loss.



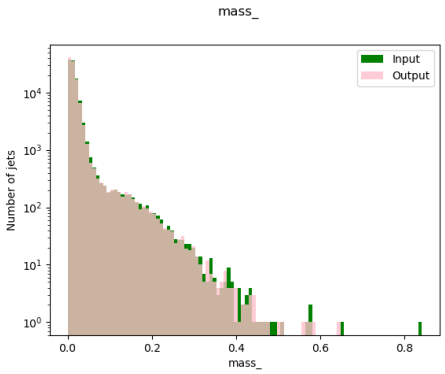
(a) AE_eta



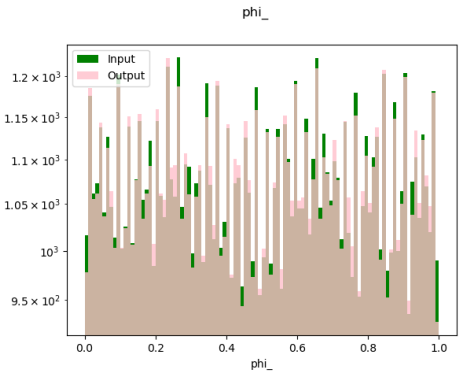
(b) SAE_eta



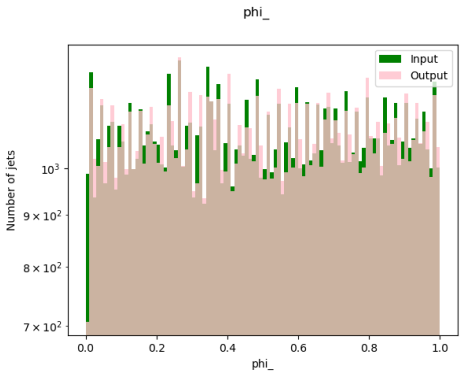
(c) AE_mass



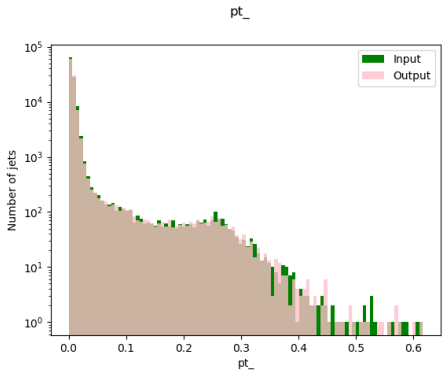
(d) SAE_mass



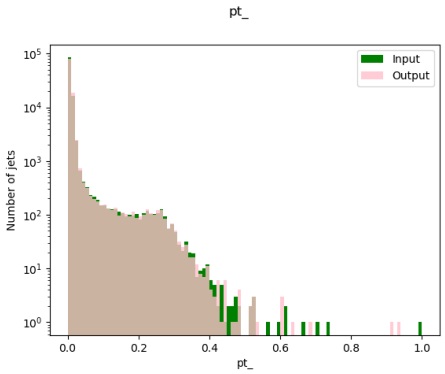
(e) AE_phi



(f) SAE_phi

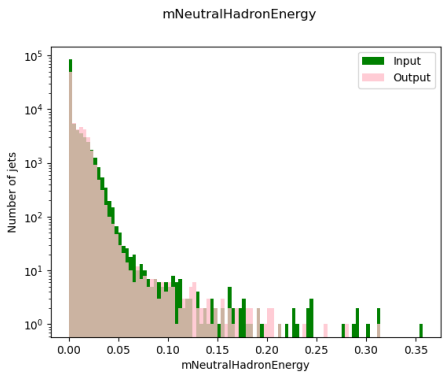


(g) AE_pT

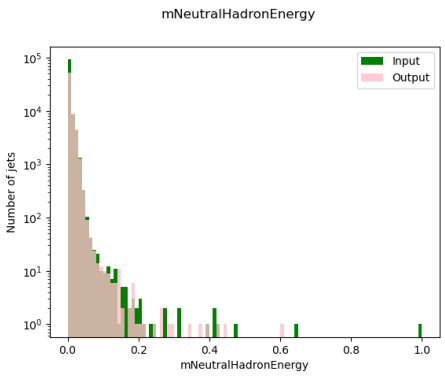


(h) SAE_pT

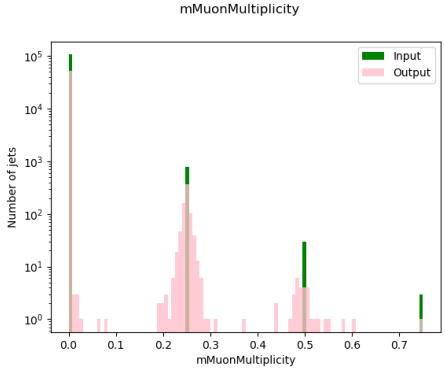
Figure 6.7: Comparison of the the AE and SAE reconstructions on the 4-momentum.



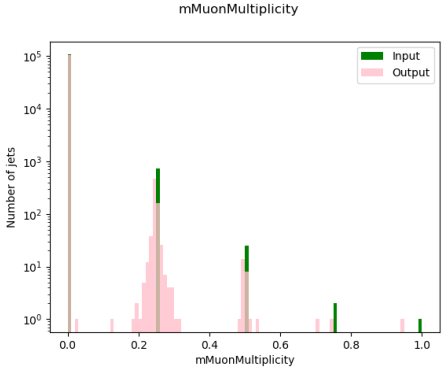
(a) AE_mNeutralHadronEnergy



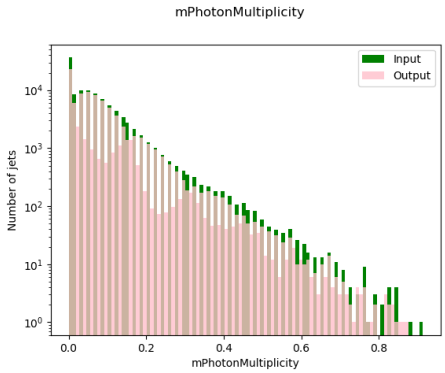
(b) SAE_mNeutralHadronEnergy



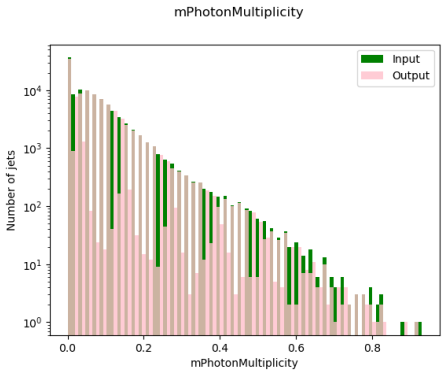
(c) AE_mMuonMultiplicity



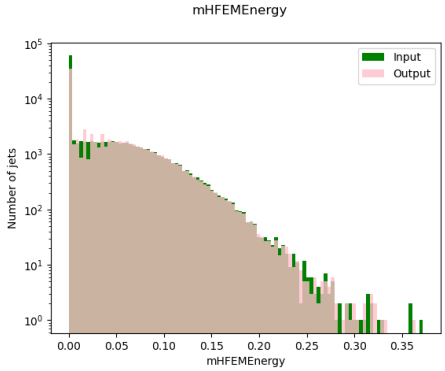
(d) SAE_mMuonMultiplicity



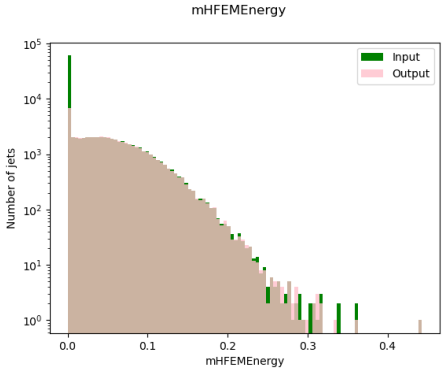
(e) AE_mPhotonMultiplicity



(f) SAE_mPhotonMultiplicity



(g) AE_mHFEMEnergy



(h) SAE_mHFEMEnergy

Figure 6.8: Comparison of the the AE and SAE reconstructions on other variables.

Chapter 7

Conclusion & Future Work

Autoencoders 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. Previous work has experimented on the basic form of an Autoencoder, and presented great performance in reconstructing the input data [5]. In this work, we initially investigated on the Standard Autoencoder and different data normalizations techniques that could improve its performance. As discussed in 6.1, scaling all the variables of the data to the same range shows to aid the Autoencoder produce better reconstructions. Subsequently, we considered and experimented on two more advanced variants of Autoencoders. These were the Variational and the Sparse Autoencoder. Moreover, two different regularization terms in the loss function of the Sparse AE were tested, the KL divergence and the L1 regularization. In summary, the Sparse AE with the L1 regularization outperformed all other networks even the Sparse AE with KL divergence, while the Variational AE presented by far the worst performance.

Future work concerns further testing of the Variational Autoencoder and in specific deeper investigation on the reason behind the KL divergence term in the loss function degrading the performance of the Sparse AE and the Variational AE. Moreover, it would be very useful to consider vizualizing on what parts of the network the Sparse AE has activated and deactivated during training, so as to further understand what input features are most important for the model. An interesting idea for the future would be to implement and experiment on a new and advanced type of AEs, the Adversarial Autoencoder [13], to investigate the ability of Adversarial learning in compressing and reconstructing trigger level data. To regularize the latent code, the Adversarial Autoencoder replaces VAE's KL-divergence with adversarial loss, where an additional discriminator component is added and the encoder will act as the generator. Finally, an important task for the ATLAS experiment at CERN is to detect anamolies in the physics data. A recent work in [14] investigates Variational Autoencoders for Anomaly detection on data produced at the Large Hadron Collider. In the future, we would like to experiment on the Sparse Autoencoder, as discussed in this work, and the Adversarial Autoencoder on task of anomaly detection, and compare our results with those in [14].

Implementation code of this project: https://github.com/Autoencoders-compression-anomaly/Deep-Autoencoders-Data-Compression-GSoC-2021.git

Bibliography

[1] CERN. The large hadron collider. https://home.cern/science/accelerators/large-hadron-collider.

[2] CERN. The atlas detector. https://home.cern/science/experiments/atlas.

[3] Wikipedia. Data compression. https://en.wikipedia.org/wiki/Data_compression.

[4] Wang, Y., Yao, H. & Zhao, S. Auto-encoder based dimensionality reduction. *Neurocomputing* **184**, 232–242 (2016).

[5] Wulff, E. Deep autoencoders for compression in high energy physics (2020). Student Paper.

[6] Wallin, E. Tests of autoencoder compression of trigger jets in the atlas experiment (2020). Student Paper.

[7] Dertat, A. Cms collaboration (2017). jetht primary dataset in aod format from run of 2012 (/jetht/run2012b-22jan2013-v1/aod). cern open data portal.

[8] Kramer, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal* **37**, 233–243 (1991).

[9] Dertat, A. Applied deep learning - part 3: Autoencoders. URL https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

[10] Weng, L. From autoencoder to beta-vae. URL https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[11] Diederik, P. K. & Welling, M. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, vol. 1 (2014).

[12] Ng, A. *et al.* Sparse autoencoder. *CS294A Lecture notes* **72**, 1–19 (2011).

[13] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I. & Frey, B. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644* (2015).

[14] Cerri, O., Nguyen, T. Q., Pierini, M., Spiropulu, M. & Vlimant, J.-R. Variational autoencoders for new physics mining at the large hadron collider. *Journal of High Energy Physics* **2019**, 1–29 (2019).