Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Theo Heimel

born in Frankfurt (Germany)

2018

The Anti-QCD Tagger

Using Autoencoders to Find Anomalies in Jet Substructure

This Bachelor Thesis has been carried out by Theo Heimel at the
Institute for Theoretical Physics in Heidelberg
under the supervision of
Prof. Tilman Plehn

Abstract

In this thesis we show how autoencoders, a type of neural networks, can be used to detect anomalous events after only being trained on QCD background events. We implement autoencoders training on two representations of the jet substructure, jet images and constituent four-vectors, using top tagging as a benchmark scenario. We discuss which preprocessing steps have to be applied to the events in both representations and how the network architectures and training parameters can be optimized. Furthermore, we show how to decorrelate the discrimination result of the autoencoder from the jet mass using adversarial networks. Finally, we test the autoencoder on samples with signal events based on models of physics beyond the Standard Model.

Zusammenfassung

In dieser Arbeit zeigen wir, wie Autoencoder, ein Typ von neuronalem Netzwerk, genutzt werden können, anomale Ereignisse zu detektieren, nachdem das Netzwerk nur mit QCD-Hintergrundereignissen trainiert wurde. Wir implementieren Autoencoder für zwei verschiedene Darstellungsweisen der Ereignisse, Jet-Bilder und Vierervektoren der Jet-Konstituenten. Wir nutzen Top-Tagging als beispielhaften Anwendungsfall, um den Autoencoder zu testen. Wir diskutieren, welche Vorverarbeitungsschritte die Trainingsdaten durchlaufen müssen und wie die Netzwerkarchitektur und Trainingsparameter optimiert werden können. Desweiteren zeigen wir, wie das Klassifikationsergebnis des Autoencoders mithilfe von Adversarial Networks von der Jetmasse dekorreliert werden kann. Abschließend testen wir den Autoencoder mit Signal-Ereignissen, die von einigen Modellen für Physik jenseits des Standardmodells vorhergesagt werden.

Contents

1 Introduction

The Standard Model of particle physics has been a great success as it provides a self-consistent explanation of the elementary particles we observed so far and three of the four fundamental forces, and many of its predictions have been confirmed experimentally. It was completed with the experimental observation of the Higgs boson in 2012 at the LHC [1, 2]. However, there are also observations like dark matter which cannot be explained in terms of the Standard Model. While it is possible that there is no physics beyond the Standard Model to be found in the energy range of the current particle accelators, it could also be that evidence for new physics is already hidden inside the vast amounts of data collected by colliders like the LHC. That makes the search for better methods for tagging, the discrimination of interesting events from the background, an increasingly important field of research.

Driven by the availability of powerful hardware and the large commercial field of application in the age of big data, there has been much progress in machine learning in recent years and many resources and powerful machine learning libraries are available to the public. Soon, the methods of machine learning were also applied to particle physics, most prominently for tagging purposes. In the area of top tagging, recent taggers based on neural networks like DeepTop and DeepTopLoLa outperform older tagging methods like boosted decision trees [3, 4].

However, these networks have to be trained on both background and signal events. Furthermore, as the classification labels of the training samples must be known, the training can only be done on Monte-Carlo-simulated events. In this thesis, we will take this one step further and develop a neural network that has to be trained only on background samples. This way, we do not even need a well-understood signal, instead, we can look for anomalies in the data. That means, our network is made to answer the question:

Is an event QCD background or not?

We will implement this anomaly detection using autoencoders, a network architecture for unsupervised learning that has the goal to find an efficient way of compressing and then decompressing its training data. When we compress the input data and then immediately decompress it using the autoencoder, we can evaluate how well the input data could be recreated from its compressed representation and use that as a measure for the quality of the compression for the given type of input data. Of course, the compression will work best for the type of data that the autoencoder was trained on. By training an autoencoder on pure background, the resulting better compression of background events compared to signal events can be used to discriminate between them.

There are two widely used approaches to machine learning in particle physics: In the image based approach, used for example in the DeepTop top tagger, methods from image recognition like Convolutional Neural Networks (CNNs) are applied to jet images which

are transverse momenta or energies of jet constituents discretized in the azimuthal angle - rapidity plane. On the other hand, in the constituent based approach, the network is directly trained on the four vectors of the jet constituents. This way, physics inspired layers like the LorentzLayer (LoLa) in the DeepTopLoLa top tagger can be added easily. In this thesis, we will implement an autoencoder for both jet images and constituents and compare their performance using publicly available Monte-Carlo-generated top tagging samples as a benchmark [5].

Often, some simple variables like the jet mass will be the dominant classification criteria but as they are not applicable in all tagging scenarios, it can be necessary to control what the network learns for a successful anomaly detection. Using adversarial networks in combination with autoencoders, we are able to impose additional conditions on our training result and decorrelate the discrimination result from certain variables. This has been done for fully supervised training in Ref. [6]. In this thesis, we present how the classification results of our autoencoder can be decorrelated from the jet mass of the events.

First, we will give some background information on the physics behind our application of machine learning and the machine learning techniques we used. Then we will explain the generation of jet images and different preprocessing methods, describe the evolution of our autoencoder architecture for images and discuss the results depending on different model parameters and preprocessing methods. We will then cover the same points for autoencoders that work directly on jet constituents. Afterwards, we will describe how we can prevent the autoencoder from learning the jet mass using adversarial training and evaluate the classification performance again. Finally, we will use the autoencoder to tag simulated events from some scenarios for physics beyond the standard model and give our conclusions and a summary of our results.

2 Physics background

2.1 The Standard Model and beyond

The Standard Model of particle physics is a theory that describes all known elementary particles (point-particles with no substructure according to today's knowledge) and three of the four fundamental interactions (electromagnetic, weak and strong interaction, but not gravitation). Figure 2.1 gives an overview over the elementary particles described by the Standard Model. The particles are categorized according to their spin into gauge bosons with spin 1, the Higgs boson with spin 0 and fermions with spin 1/2. The gauge bosons are mediators for the fundamental interactions: The photon mediates the electromagnetic force, the gluon mediates the strong force and the W^+, W^- and Z^0 bosons mediate the weak interaction. The Higgs boson is needed to explain why the W^+, W^- and Z^0 bosons – other than photons and gluons – have non-zero masses. The fermions split up into leptons and quarks. These are organized into three generations of matter ordered by increasing mass. The elementary particles from the first generation, electrons, neutrons and protons, are the building blocks of the matter surrounding us, whereas the higher generations form particles with short lifetimes (or no particles at all in case of the top quark) that can only be observed after collisions of high-energetic particles, for example the cosmic radiation or in particle accelerators.

The predictions of the Standard Model match the experimental results very well, however, there are 19 free parameters that have to be determined from measurements. But there are also observations that call for physics beyond the Standard Model. For example, neutrino oscillations occur and hence, neutrinos must have non-zero mass [8]. Also, dark matter and dark energy that were observed on cosmological scales cannot be explained with the elementary particles of the Standard Model.

2.2 Particle colliders

Particle colliders are built to create heavy particles with short lifetimes. The energy needed to form those particles is provided by the high kinetic energy of the colliding particles. Various kinds of particles can be accelerated, for example (anti-)protons, electrons, positrons or heavy ions. Other important parameters are its center of mass energy $\sqrt{s}$ that determines which particles can be created during the collision and the number of accelerated particles per bunch in the beam that - together with the total cross-section - determines the luminosity of the collider.

The current largest particle collider is the LHC, a proton-proton collider that has a center of mass energy of up to $\sqrt{s} = 14TeV$. This is enough to form sufficiently high numbers of Higgs bosons and consequently, the first experimental observation of the Higgs boson is considered the greatest success of the LHC. While confirming the predictions of the

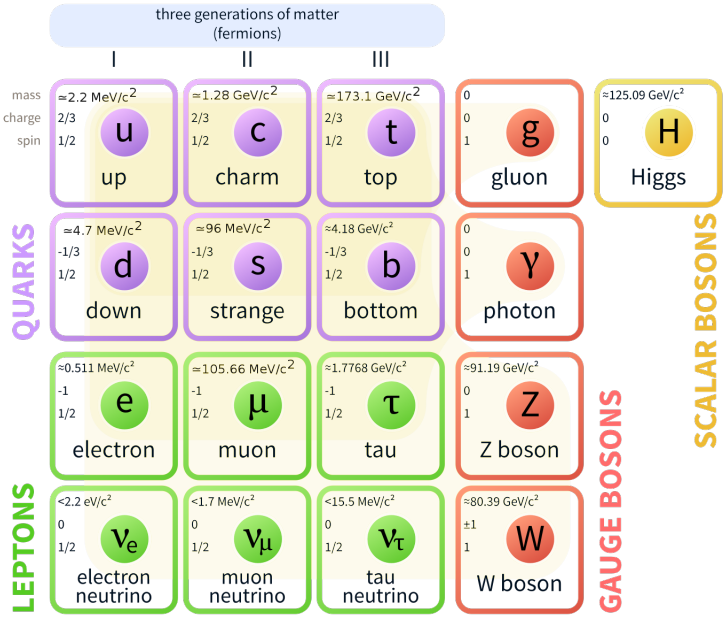


Figure 2.1: The elementary particles in the standard model. Taken from [7].

Standard Model with high precision, no evidence for physics beyond the Standard Model has been found at the LHC until now.

The points of collision in the accelerator surrounded by detectors (e.g. calorimeters), so the direction, momentum and energy of particles produced during and after the collision can be reconstructed and the four-momenta of these particles can be given. In case of the LHC, there are four large detectors: ATLAS, CMS, LHCb and ALICE.

2.2.1 Jets

When quarks or gluons with high energies are created in a collider, they undergo the process of hadronization. Because of color confinement, quarks and gluons cannot exist individually but only in a bound state with neutral ("white") color charge (i.e. between three quarks with different colors or two quarks with a color and its corresponding anti-color). The potential energy between two quarks increases rapidly with their distance, so it becomes energetically favorable for them to form hadrons with quark-antiquark pairs created from the vacuum. Before the quarks finally form a bound state, this process of creation of quark-antiquark pairs can happen multiple times. For further analysis, the final hadrons have to be recombined again using jet clustering algorithms like anti-k_T [9]. The outputs of these algorithms are referred to as jets. Jets that are created by the hadronization of heavy particles and have a large geometrical size, are called *fat jets*. Jets caused by QCD processes during which no interesting particle we consider "signal" is formed, are summed up under the term *QCD background*.

2.2.2 Top quarks

With a mass of 175 GeV, the top quark is the most massive quark in the Standard Model and consequently it has a very short mean lifetime of $5 \cdot 10^{-25}$s. Because of this, the top quark decays before it can hadronize. The most common process for top quark production is the formation of a top anti-top pair through the strong iteraction. The top quark decays into a W-boson and a bottom, strange or down quark. The W-boson can decay into a lepton and a neutrino or into two quarks. The quarks produced during the top decay have much longer lifetimes than the top quark and thus hadronize and form jets.

2.2.3 Quantifying particle and jet properties

Consider the four-momentum p of a jet constituent and orthogonal spatial coordinates where the z-axis is the beam axis. The transverse momentum p_T is then defined as

$$p_T^2 = (p^x)^2 + (p^y)^2 \tag{2.1}$$

The polar angle, θ, is the angle between the three-momentum $\vec{p}$ of the particle and the beam axis. The azimuthal angle, ϕ, is the angle between the three-momentum projected to the x-y-plane and the x-axis. The *pseudorapidity* is a measure for the polar angle of a particle relative to the beam axis. It is defined as

$$\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right) \tag{2.2}$$

ϕ and η are useful quantities to express the direction of a particle. Together with energy $E = p^0$ or the transverse momemtum p_T, they can be used to generate jet image. These images will be used as input for the autoencoder in this thesis. Another important quantity is the angular separation $\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$ between two particles where $\Delta\eta$ and $\Delta\phi$ are the differences in pseudorapidity and azimuthal angle. The invariant mass, m, of a particle can be calculated from its four-momentum by contracting it with the Minkowski metric: $m^2 = \eta_{\mu\nu}p^\mu p^\nu = E^2 - (p^x)^2 - (p^y)^2 - (p^z)^2$

The same quantities can also be determined for the whole jet by summing the jet four-momenta, p_j, for all n constituents and then using the same definitions as above:

$$p_{jet}^\mu = \sum_{j=1}^{n} (p_j)^\mu \tag{2.3}$$

3 Machine learning background

Machine learning describes the process when a computer "learns" to do a specific task from a given dataset rather than being explicitly programmed for that task. One differentiates between supervised learning where the expected outputs for the training dataset are given and unsupervised learning where this is not the case.

In this thesis, we base our machine learning programs on artificial neural networks. They consist of simple units – often called neurons – which have multiple inputs and perform a mathematical operation on those. The neuron's inputs and outputs are connected via trainable weights which are adjusted by an optimizer to train the network. The neurons are often organized into layers with inputs coming from the previous layer and outputs going to the next layer. The layers can also have functions that serve a more specific purpose, for instance, they can be designed for image recognition. These concepts will be more thoroughly explained in the following sections.

3.1 Types of layers

Dense layers: Each of the n outputs of the layer is connected to all m inputs by multiplying them by a weight and then summing the results up. Hence, there are $n \cdot m$ trainable weights. By arranging them into a $n \times m$ matrix w_{ij}, the function of the layer can be expressed as $y_i = b_i + \sum_j w_{ij} x_i$. b_i is the bias of the layer, which is also a vector of trainable parameters. x_i are the inputs of the layer. Network architectures containing only dense layers are called fully connected.

Convolutional layers[1]: This type of layer takes multiple two-dimensional images as input (called feature maps) and generates the output pixels by performing an elementwise multiplication of a $k \times l$ matrix which remains the same for the entire image – the convolution kernel – with parts of the image of the same size and then summing the results up. By iterating over the whole input image, a two-dimensional output image is created. This process is called a two-dimensional discrete convolution and is repeated for all pairs of input and output feature maps and the results for each output feature map are summed up. For n input feature maps and m output feature maps, there are $n \cdot m \cdot k \cdot l$ trainable parameters. Convolutional layers are often used in image recognition, because they are translation invariant by construction. Similar structures in different parts of an image are treated in a similar way by these layers.

Average Pooling[1]: Pooling layers are defined for two-dimesional input data with size $n \times m$ and are characterized by the size $k \times l$ of their pooling windows. The input data is divided into cells of size $k \times l$, then the average value for each of these cells is calculated. The resulting output size is $n/k \times m/l$. If receiving multiple feature maps from a convolutional layer, the pooling is applied to all of them seperately.

Maximum Pooling[1]: This layer is similar to Average Pooling, with the only difference being that the maximal value of the units in the pooling windows is taken instead of averaging over them.

Upsampling[1]: Each row of the input data is repeated k times, then all columns are repeated l times such that for an input input size of $n \times m$, the output size is $nk \times ml$.

Flatten and **Reshape**: These layers only change the shape of the data, for instance by increasing or decreasing (flattening) the dimensionality while keeping the total number of units constant.

Convolutional layers and pooling or upsampling layers are often used in conjuction: One or more convolutional layers are followed by an upsampling or pooling layers, again followed by convolutional layer. Neural networks composed of these layers are called Convolutional Neural Networks (CNN) and are often used in image recognition because of the translation invariance of the convolutional layers.

To describe network architectures in this thesis, we introduce the following notation for the layers:

- *Dense(n)*: Dense layer with n output units.
- *Conv2D(n,m)*: Convolutional layer with n output feature maps and a $m \times m$ convolution kernel.
- *AveragePooling2D(m)*: Average pooling layer with a $m \times m$ pooling window layer. Analog for MaxPooling.
- *UpSampling2D(m)*: Upsampling layer with $m \times m$ upsampling factor.

3.2 Activation functions

Most types of layers like dense and convolutional layers perform linear operations on their input data. However, if the network should be able to learn more complex, non-linear relations between input and desired output, non-linearities have to be introduced. This is usually done by applying a so called *activation* function to the output of each layer. The simplest activation function is *linear activation* which does not change the output at all. Examples for non-linear activation functions used in machine learning are the sigmoid function $S(x) = e^x(e^x+1)^{-1}$ or the tangens hyperbolicus function. Both of these functions have the problem that their gradients vanish for very high or low inputs, so in the case of bad initialization values for the weights, it can be hard for the optimizer to escape from an always-firing or always-zero state. This problem is (partially) solved by ReLU (rectified linear unit) activation, which is defined as

$$\mathrm{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.1}$$

This activation function makes sparse outputs possible because the output is zero for all negative input values. The gradient of this function is vanishing for negative x, so dead

[1] These layers are described for two-dimensional data here, however they can be generalized to an arbitrary number of dimensions.

neurons (i.e. neurons never outputting anything different from zero because of inappropriate initialization) can still occur. That is not the case for the LeakyReLu function defined as

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.2}$$

with a parameter $0 < \alpha < 1$. A variation of LeakyReLU is PReLU (parametric rectified linear unit) activation, where α is a trainable parameter. An example for a loss function that operates on the whole output vector of a layer (with n entries) and not only on single values is SoftMax activation, where the summed output of the layer is normalized to 1.

$$\text{SoftMax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3.3}$$

3.3 Training

Before the training of a neural network can start, the trainable weights have to be initialized. There are many possible ways to do the initialization, from trivial ones (setting all the weights to zero) to more sophisticated ones (taking into account the numbers of inputs and outputs to a layer). In this thesis, we will always use the initialization method proposed in Ref. [10] called Glorot uniform initialization. For each layer, the weights are drawn from a uniform distribution within the range $[-A, A]$, where $A = \sqrt{6/(n_{in} + n_{out})}$ with the number of input and output units of the layer n_{in} and n_{out}.

The actual learning process of the neural network is performed by the optimizer which iteratively minimizes the so called *loss function*. This is calculated from the network input and output. Some common loss functions will be described in the following section. The training process is split up into *epochs* where in each epoch, the full training sample is fed through the network once. The loss function is evaluated for these samples and the optimizer then tries to minimize it by changing the trainable weights in the model. Most optimizers are derived from the *gradient descent* algorithm, which evaluates the gradient of the loss at the current state and then changes the weights in the direction of the steepest descent. The gradient is calculated using the backpropagation method. A commonly used optimizer is SGD (Stochastic Gradient Descent), where the gradient is only evaluated for a random subset of the trainable parameters in order to reduce the time needed for every optimizing step. An important parameter for this type of optimizer is the *learning rate* which determines how much the weights change in one epoch. Training with a constant learning rate has the problem that the step size around the optimal weights may be too large and the algorithm is thus oscillating around the optimal results. To prevent this from happening, more sophisticated optimizers, such as Adam [11], adapt the learning rate during training. To get a better parallelization of the learning, for example by using GPUs, the training sample is split up into smaller *batches*.

3.4 Autoencoders

Autoencoders are a type of model used for unsupervised learning. Their goal is to learn an efficient compression for arbitrary input data. This is achieved by building a model that has input and output layers of the same size as the training data. The model consists of layers with a decreasing number of neurons until a layer with the lowest number of neurons is reached. These layers form the so called *encoder part* of the autoencoder. They are followed by more layers, this time with an increasing number of units, also called the *decoder part*. The network is then trained to minimize the loss between input and output, such that the decoder part behaves inverse to the encoder part. The central layer with the minimal number of units will be called *bottleneck* in the further course of this thesis. For autoencoders, arbitrary types of layers can be used, so both convolutional and dense layers are suitable. However, for the most part it is advisable to use layers whose function can easily be inverted such that the decoder part of the network can be built symmetric to the encoder part.

An important model parameter that determines how strong the compression of the network will be, is the *bottleneck size*, i.e. the number of neurons in the bottleneck layer. If the bottleneck size is decreased, usually the loss increases, because the input cannot be reproduced as good as before.

3.5 Metrics for the network performance

3.5.1 Loss functions

A loss function compares the output of a neural network with the desired result, yielding a scalar value that is minimal when the desired ("true") and predicted output of the network match perfectly. A simple example for a loss function is the Mean Squared Error (MSE) defined by

$$L_{MSE}(y_{true}, y_{pred}) = \frac{1}{N^2} \sum_{i=0}^{N} (y_{true} - y_{pred})^2 \tag{3.4}$$

There are also widely used loss functions based on information theory like the categorical crossentropy, which is defined as follows for binary classification problems as

$$L_{CE}(y_{true}, y_{pred}) = -y_{true} \log(y_{pred}) - (1 - y_{true}) \log(1 - y_{pred}) \tag{3.5}$$

It can be generalized for higher numbers of classes. However, crossentropy is only suitable for classification problems and not for autoencoders, hence we use MSE loss to train the autoencoder. In the further course of this thesis, multiple variations of the MSE loss function, especially for images, will be examined.

3.5.2 ROC curve

Samples for binary classification can be assigned to one of two classes. The one-dimensional output of the classifier is compared to a threshold value to get a prediction for the class of a sample. Receiver operating characteristic (ROC) curves are graphs where the true positive rate (TPR, also called sensitivity or signal efficiency) is plotted against the false positive rate (FPR) or the background rejection, which is the inverse false positive rate, to illustrate the discrimination power of a binary classifier for the whole range of possible threshold values. To create a ROC curve, the true class of each sample and the predicted, continuous classification score have to be known.

The area under the ROC curve (AUC) that can be calculated by integrating the TPR over the FPR range $[0, 1]$, is a useful metric for the performance of a classifier: If there is no correlation between the class of a sample and its classification score, the TPR and FPR are equal for every threshold setting and the resulting AUC is 0.5. The higher than 0.5 the AUC, the better is the classification performance. For AUCs smaller than 0.5, there is also some classification power, however the classification score has to be reversed to get useful results.

The AUC has the disadvantage that the parts of the curve with FPRs and TPRs near 0 and 1 respectively have a large influence on the AUC score, although they are not relevant for any practical use case of the classifier. Thus, the AUC is often not the best metric for the performance of a classifier and it is useful to give the FPR at the desired TPR.

4 Autoencoder for jet images

4.1 Sample set

All the tests with the autoencoder in this section were performed using the *Top Tagging Reference Dataset* that is based on the dataset used in the DeepTop paper [5, 3]. It consist of 1,200,000 training events, 400,000 testing events and 400,000 validation events, each with 50% signal events and 50% background events.

The production of top quark pairs and QCD dijets is simulated at a collider energy of $\sqrt{s} = 14$ TeV using Pythia 8.2.15 [12], a Monte Carlo event generator that computes the decay and hadronization products for the processes of interest. The top events are considered signal and the QCD events are considered background. The detector simulation is then done using Delphes 3.3.2 [13] with a standard ATLAS detector card. Delphes simulates the detection properties and limitations of real-world particle detectors like ATLAS. The p_T range of the events is limited to $550 \ldots 650\,GeV$ and the jets are required to have $|\eta_{jet}| < 2$. Afterwards, the jet clustering is performed with FastJet 3.1.3 [14] using the anti-k_T algorithm [9] with a jet cone size of $\Delta R = 0.8$.

The four-momenta for up to 200 of the jet constituents are stored, ordered by decreasing p_T. For jets with less than 200 constituents, the empty four-momenta are filled with zeros. All events are flagged whether they are top or QCD events, such that the classification result can be evaluated.

4.2 Image generation and preprocessing

4.2.1 DeepTop preprocessing

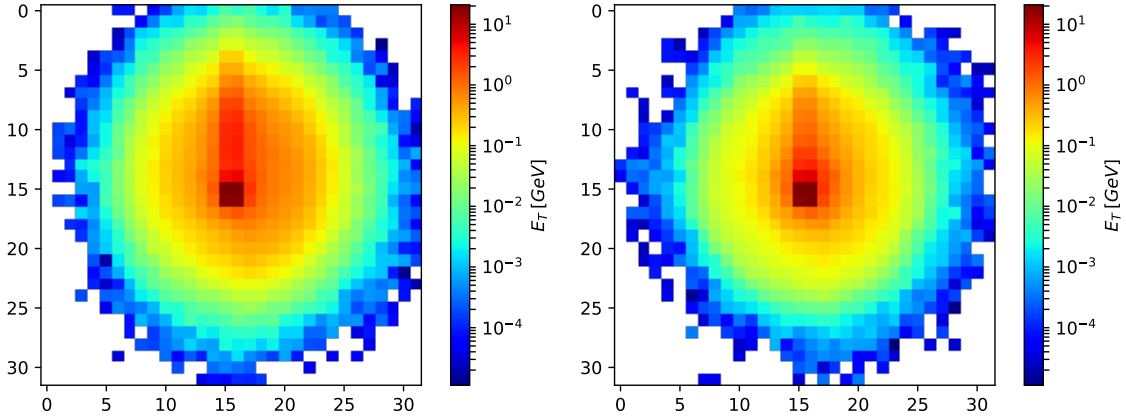
The autoencoder for jet images is based on two-dimensional Convolutional Neural Networks, so images representing the jet data have to be generated. Like for the DeepTop CNN [3], the pseudorapidity η and the azimuthal angle ϕ are calculated for each jet constituent. Then, a $n \times n$ lattice is set up for the (η, ϕ) plane with values for ϕ between -180 and 180 degrees. For each jet constituent, the corresponding pixel in the lattice is assign the energy of the constituent as intensity. The range for η has to be selected such that most jet constituents can be placed inside the lattice.

Afterwards, up to four preprocessing steps are applied:

- *Centering*: The images are translated such that the pixel with the highest energy is always in the center of the image.
- *Rotating*: The images are rotated such that the pixel with the second highest intensity is above the central pixel.

Final image size	Zoom factor	Lattice size n
24	1.0	55
32	1.0	72
40	1.0	90
40	1.5	135
40	2.0	180

Table 4.1: Settings for the lattice size n for some final image sizes and zoom factors



(a) Signal events

(b) Background events

Figure 4.1: Mean jet images for signal and background events with image size 32×32 and full preprocessing applied. (Means taken over all validation samples)

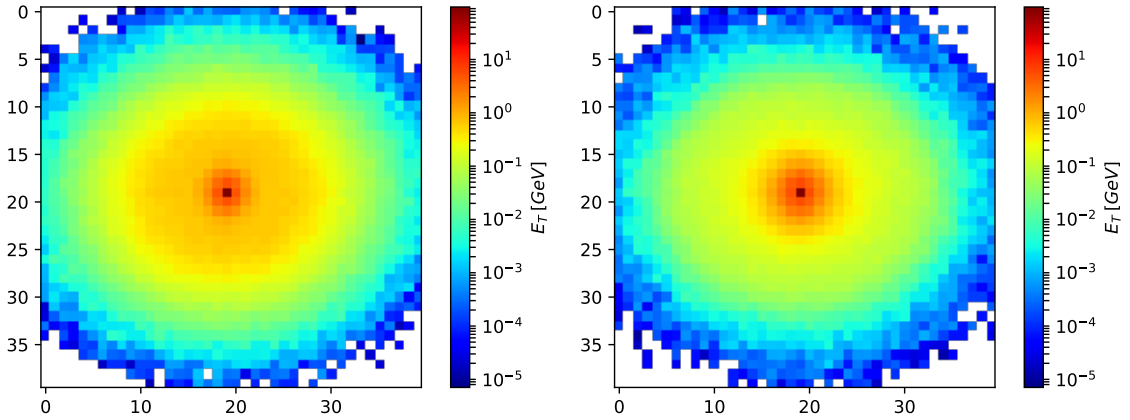
- *Flipping*: The images are flipped along the y axis such that the pixel with the third highest intensity is always on the right side of the image.
- *Cropping*: The images are cropped to the desired size.

Centering and cropping were always applied whereas images with rotation and flipping turned on ("full preprocessing") and off ("minimal preprocessing") were generated. $[-2.6, 2.6]$ turned out to be a reasonable choice as range for η. Images with different final sizes (after cropping) were generated. Some were zoomed in such that not all constituents were visible in the final images. The appropriate settings for these variants are shown in table 4.1.

The results for full and minimal preprocessing and two different image sizes can be seen in Figures 4.1 and 4.2. Figure 4.3a shows the numbers of active pixels for signal and background events. Because of the limited resolution introduced by pixelating, the distributions peak at lower numbers compared to the distribution for the number of jet constituents shown in Figure 4.3a.

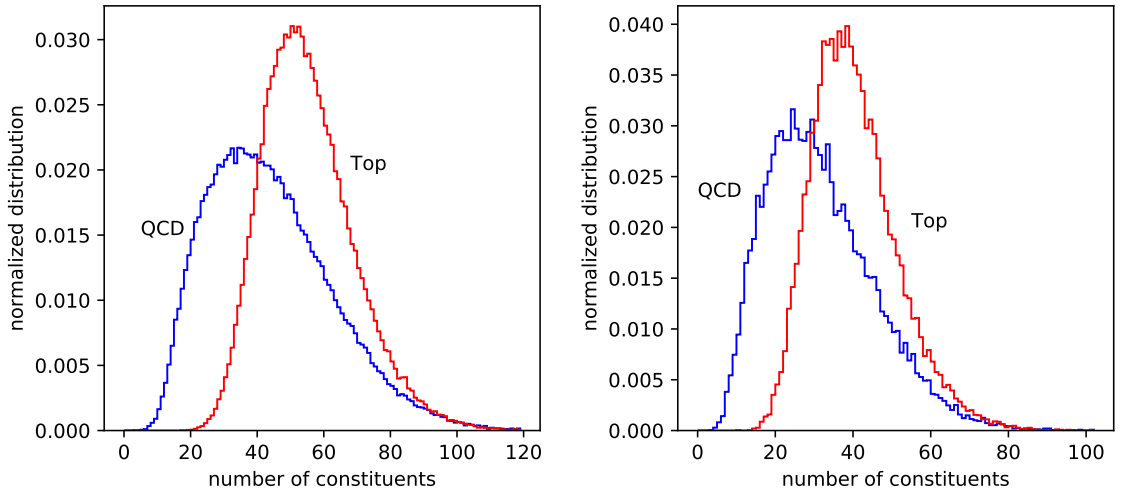
4.2.2 Improved preprocessing

Sebastian Macaluso and David Shih proposed an improved preprocessing [15] method, where they did the preprocessing steps before pixelating the image and used p_T as pixel intensities instead of E_T: First, they calculated the centroid of the jet in the (η, ϕ) plane weighted by p_T. Then, they applied a shift such that the centroid is at the origin and



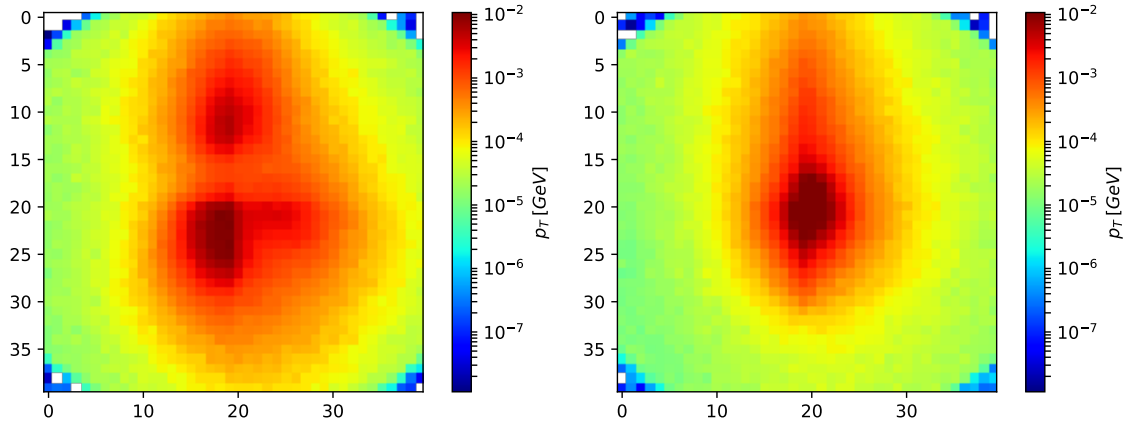
(a) Signal events (b) Background events

Figure 4.2: Mean jet images for signal and background events with image size 40×40 and minimal preprocessing applied. (Means taken over all validation samples)



(a) Number of constituents (b) Number of pixels

Figure 4.3: Histograms of the number of constituents and pixels different from zero after improved preprocessing for 40×40. The histograms were calculated using all 400,000 events of the validation sample of the Top Tagging Reference Sample.



(a) Signal events

(b) Background events

Figure 4.4: Mean jet images for signal and background events with image size 40×40 and improved preprocessing applied. (Means taken over all validation samples)

performed a rotation such that the major principal axis is vertical. Next, they flipped the image along both axes so that the maximum is in the upper right quadrant. Only then, the image is pixelated by dividing up the region $[-0.57, 0.57] \times [-0.69, 0.69]$ into a 40×40 grid and summing up the p_T values of jet constituents covered by those pixels. Afterwards, the images are normalized by dividing through the total p_T. The resulting mean jet images can be seen in Figure 4.4. The peak in intensity for the W^+ boson generated during the top decay is now clearly visible in the mean signal image. That was not the case for the DeepTop preprocessing (see Figure 4.1).

4.3 Training and testing setup

We implemented the autoencoder setup using the Keras library [16] with a Theano backend [17]. We used the Adam optimizer [11] to train the network. The batch size was kept at 128 for most of the tests as varying had no impact on the performance. After each epoch, a testing loss was calculated using the testing samples. The maximum number of epochs was set to 200 with early stopping after 10 epochs enabled, i.e. when the testing loss did not improve for 10 epochs, the training stopped. Unless otherwise stated, the mean squared differences loss function was used. Both purely convolutional networks and networks with dense layers in the middle were examined.

Finally, the discrimination power of the autoencoder was tested using the (mixed) validation samples. This was done by applying the same loss function as for the training to both background and signal samples. The result was used as classification score. Because the network was trained on background samples, lower losses are expected for these events. Using this score, a ROC curve, the corresponding Area Under Curve and the FPR at a given TPR (here: 0.5) were calculated. Additionally, mean images of the input and output data and the mean squared difference between them were generated to visualize how well the input images could be reproduced by the autoencoder.

4.4 Network architecture

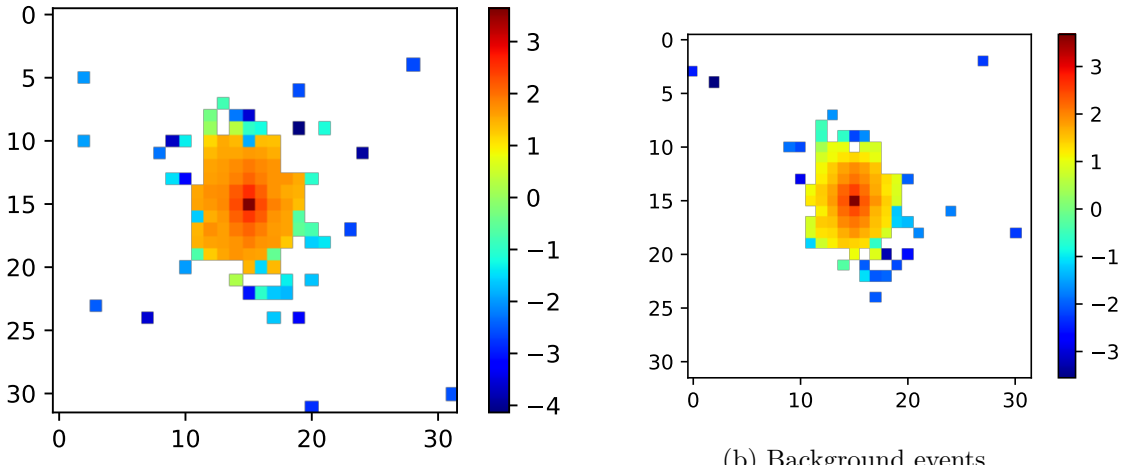
4.4.1 Finding the optimal network architecture

We began with a simple model containing only Conv2D and AveragePooling layers in the encoder part and Conv2D and Upsampling layers in the decoder part. First, we used fully preprocessed 32×32 images. We first focused on decreasing the loss for background events in order to get into a regime where the loss of the signal events does not decrease accordingly, so the discrimination power will get better. It is important to notice that the bottleneck size should be kept constant during that process. This is because we want the autoencoder to find an efficient compression, that focuses on the distinct properties of QCD jets. If we increase the bottleneck size, we can yield better results in terms of the loss function, but we expect the network to focus on more general properties of jets that also apply to top jets. Hence, we expect the loss for signal events to also decrease, which would lead to a worse discrimination result.

We found that the interchange of AveragePooling and MaxPooling layers had little to no effect on the classification performance, neither for simple nor for more sophisticated models. The number of convolutional layers between the pooling layers had a much larger impact on the classification result: We saw an improvement for two convolutional layers before each pooling layer. However, higher numbers of convolutional layers do not show further improvement. Also, the effect of the convolution kernel size on the result has to be considered. For 32×32 image resolution, we saw that the kernel size should not be smaller than 4×4 and for 40×40 images, 5×5 or 6×6 are appropriate choices. Furthermore, the choice of feature maps for the dense layers influences the performance. For the outer convolutions, it should be at least 10 and should be reduced towards the bottleneck. Even larger numbers of feature maps do not yield significant improvements. In Ref. [15], numbers of feature maps up to 128 were used, but that did not lead to any improvements in the case of the autoencoder while slowing down the training because of the higher number of trainable weights.

For purely convolutional networks, i.e. no layers other than convolutions and poolings were present in the model, good classification results for more than 2 pooling layers and thus a bottleneck size of one sixteenth of the number of input pixels were not possible. As a consequence, we also introduced dense layers in the middle of the network for the bottleneck. This way, we could combine the advantages of convolutional layers for image recognition and the flexibility of dense layers. This approach has several advantages: Because the jet images clearly have regions with different functions in case of preprocessed images, it is clear that it is harder for translation-invariant layers like convolutions to reproduce these distinct parts of the image for large compression factors. Also, the bottleneck size can be chosen freely, because the architecture was not limited to reducing the number of pixels by powers of two. An interesting observation is that the network started to focus more on the center of the image once we introduced dense layers while keeping the output intensities very small in other parts of the image. This can be seen in Figure 4.5. The central pixels have the highest mean intensities, hence they also contribute to the loss the most, so it is reasonable for the network to focus on these pixels.

When introducing the dense layers, the last convolutional layer of the encoder part should not be followed by a pooling layer and the first convolutional layer of the decoder part should not be preceded by an upsampling layer, because the dense layers are able to find a



(a) Signal events

(b) Background events

Figure 4.5: Mean autoencoder output images for signal and background events for a model containing dense layers. The intensity scale shows the energy and is to be understood logarithmically. (Means taken over all validation samples)

way of up- or downsampling that is more efficient in preserving the information than the very simple strategies used by pooling and upsampling layers. Also, the number of dense layers has an impact on the results: The performance was best if the number of dense layers was low, that is, the progression of the number of units was very steep in the center of the network. Four dense layers proved to be a reasonable choice. For larger numbers of dense layers, we found the training stability to be reduced and the discrimination performance was slightly decreased.

Another important aspect of the model to consider is the activation function. First, we used ReLU as activation for all layers, but replacing ReLU with PReLU significantly improved the discrimination result. For other activation functions like tanh and sigmoid, the performance was much worse than for conventional ReLU. The most plausible explanation for the issues with some activation functions is the vanishing gradient problem described in section 3.2. The last layer of the network has a special role because it must be able to reproduce the input image. As the pixel intensities of the images are never below zero and the images are sparse, we chose ReLU as activation function for the last layer, because its output is always greater or equal to zero and it is not difficult for the network to achieve sparse output images because the whole negative input range is mapped to zero by this activation function.

The influence of the batch size on the classification result is small. We had good results when the batches contained 128 events. For much larger or much smaller numbers, the performance of the network, the training time per epoch or the training stability were impaired. To reduce the time needed for the training, the number of training and testing samples can be reduced. We had good results for 100,000 training samples and 50,000 testing samples, because the stability and classification power are not reduced for that choice.

The choice of preprocessing proved to be one of the most important steps to obtain a good discrimination performance. First, we used the full preprocessing that was also used for DeepTop but we were not able to get beyond an area under ROC curve of 0.80 by training on those samples. There are a few reasons for the bad performance of this preprocessing

method. Firstly, all the preprocessing steps were done after the pixelization but especially operations like rotations are hard to perform accurately at the low resolutions used for the images. Moreover, the translation, rotation and flipping preprocessing operations are based on the three hardest pixels, so there can be a large difference between the preprocessing results between jets with few and very many constituents. Also, if the intensities of the three hardest pixels are very similar, the preprocessing operations possibly introduces fluctuations that are hard for the network to reproduce.

The network performance can be improved up to an AUC of 0.85 by using minimal preprocessing instead, even though the number of degrees of freedom is larger in that case because no symmetries are removed by the preprocessing. When using minimal preprocessing, we found that the best image resolution to be 40×40. However, the training with minimal preprocessing has severe stability problems because the central pixel dominates the loss function. The easiest solution for these problems is to ignore the central pixel in the calculation of the loss function. The discrimination power of the autoencoder is slightly reduced by that method. A better way is to apply a cut to the intensity of the central pixel. The convergence problems of the network are mainly caused by a small part of the training samples with very high intensities of the hardest pixel. Thus, the maximum intensity of the pixel is set such that for a certain percentage of the training samples, the energy of the central pixel is cut and set to the maximal energy of the other samples. We had the best results when we cut the energy for 10% of the training sample which leads to a stable performance with a maximal discrimination power of AUC 0.85.

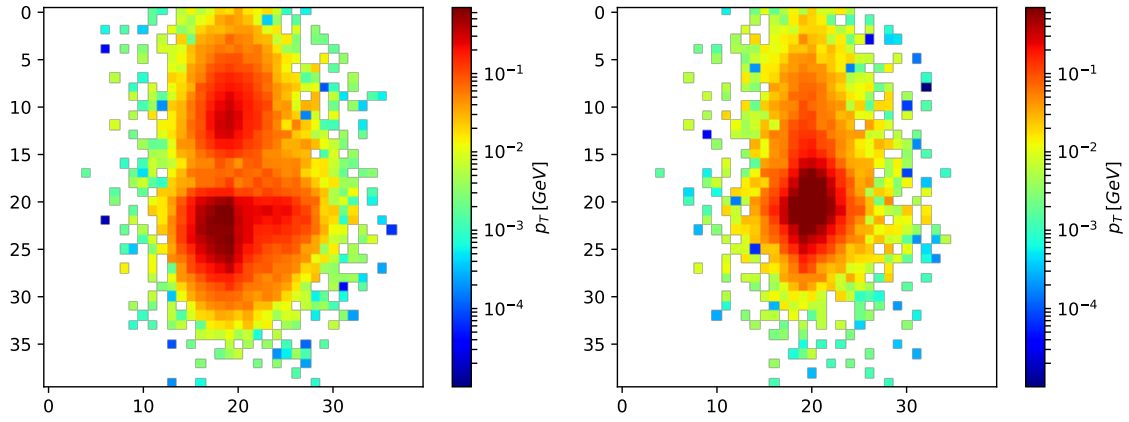
We had even better results using the improved preprocessing proposed in Ref. [15]. Here, the problems of the DeepTop preprocessing explained above are solved by doing all preprocessing steps *before* pixelization and relying on properties of the whole image like the p_T-centroid and the major principle axis of the images instead of using the three hardest pixels. Indeed, the performance increased up to an AUC of 0.89. But the improved preprocessing had stability issues too because sometimes the training stopped with no active output pixel at all. This was solved by changing the activation function of the last layer to linear activation. Like that, the sparsity of the output was no longer enforced (which proved not to be a problem) but it was also impossible for the network to easily set all pixels to zero because the local minimum of the loss function was removed. The mean output images of the autoencoder for images with improved preprocessing can be seen in Figure 4.6.

4.4.2 Final network architecture for images

Taking into account the considerations from the previous section, the architecture for which we had the best results is as follows (using the notation introduced in 3.1):

Conv2D(10,5) $\rightarrow$ *Conv2D(10,5)* $\rightarrow$ *AveragePooling2D(2)* $\rightarrow$ *Conv2D(5,5)* $\rightarrow$ *Conv2D(5,5)* $\rightarrow$ *Flatten* $\rightarrow$ *Dense(100)* $\rightarrow$ *Dense(32)* $\rightarrow$ *Dense(100)* $\rightarrow$ *Dense(400)* $\rightarrow$ *Reshape* $\rightarrow$ *Conv2D(5,5)* $\rightarrow$ *Conv2D(5,5)* $\rightarrow$ *UpSampling2D(2)* $\rightarrow$ *Conv2D(10,5)* $\rightarrow$ *Conv2D(1,5)*

Figure 4.7 is a drawing of this architecture. All convolutional and dense layers have PReLU activation functions, except for the last layer that has ReLU activation for the minimally preprocessed sample or linear activation for the samples with improved preprocessing. The input images are padded with zeros before the convolution is applied, so that the images do not get smaller when calculating the convolution and the size of the input and output image is always the same.



(a) Signal events

(b) Background events

Figure 4.6: Mean autoencoder output images for signal and background jet images generated with improved preprocessing. (Means taken over all validation samples)

As input data, we used either minimally preprocessed samples or samples with improved preprocessing, both with a size of 40×40. In both cases, we only used a subset of 100,000 events of the Top Tagging Reference Sample so that the network can be trained in a reasonable amount of time. To prevent instabilities of the training with minimally preprocessed images, we limited the energies of the central pixel such that it was cut for 10% of the events. That corresponds to the energy $E = 297\text{GeV}$. The classification performance is always evaluated using the full validation sample.

We used MSE as defined in equation 3.4 as loss function. For minimally preprocessed images, we trained the network up to 200 epochs, but stopped the training if the testing loss did not improve for 10 epochs. For the improved preprocessing, shorter training times were enough, so we used a maximum of 100 epochs and again applied earlystopping after 10 epochs. The batch size was set to 128 in both cases. As optimizer we used Adam with the learning rate 0.001.

4.5 Tests and results

In this section, we will discuss quantitative results for the performance of the image autoencoder with an architecture as described in the previous section.

4.5.1 Comparison of minimal and improved preprocessing

To compare the performance of minimal and improved preprocessing, we trained the same model as described in the previous section with both types of samples. The bottleneck size was 32. The only difference between the models was the activation function of the last layer, that was ReLU for minimal and linear for improved preprocessing. The training configuration was also as described above. As seen in Figure 4.8, the background rejection was higher for the improved preprocessing for all relevant parts of the ROC curve. The area under curve was 0.84 for minimal preprocessing and 0.87 for improved preprocessing.

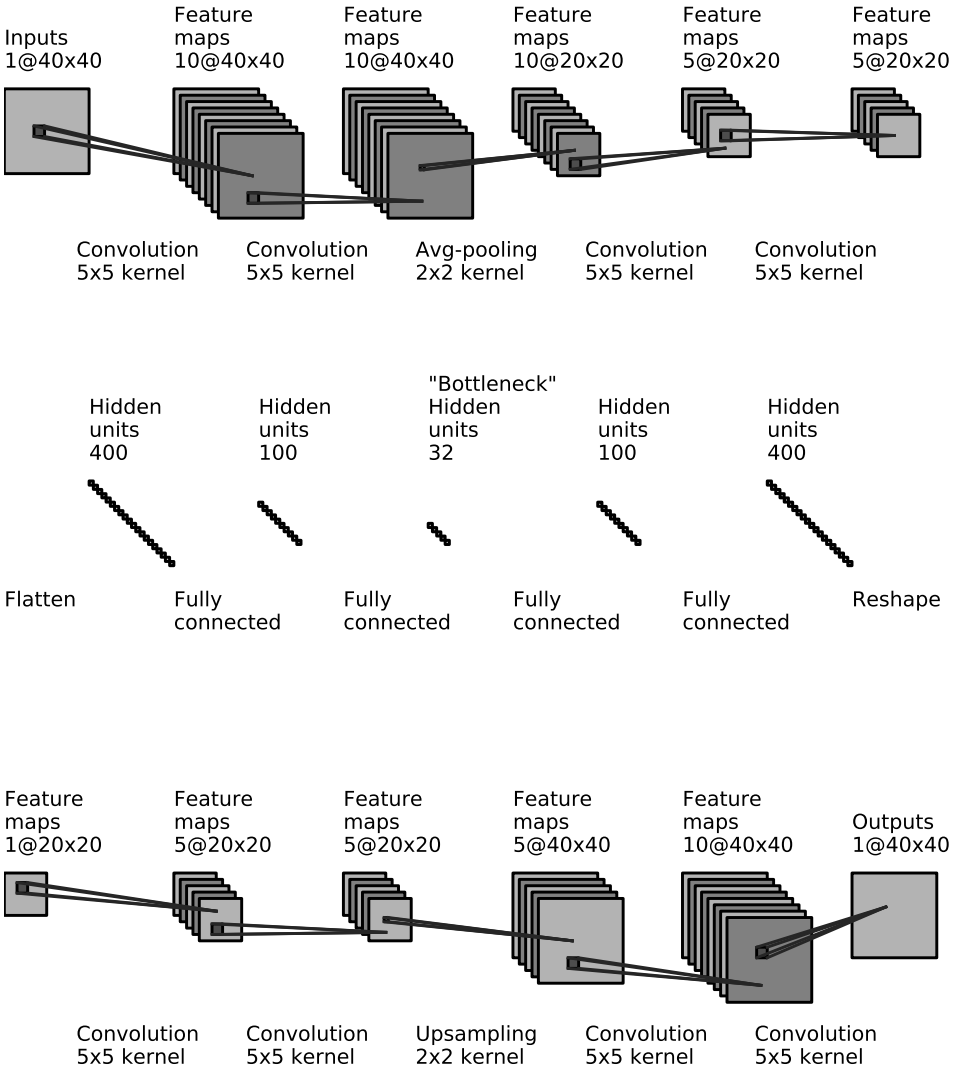


Figure 4.7: Final architecture of the image autoencoder.

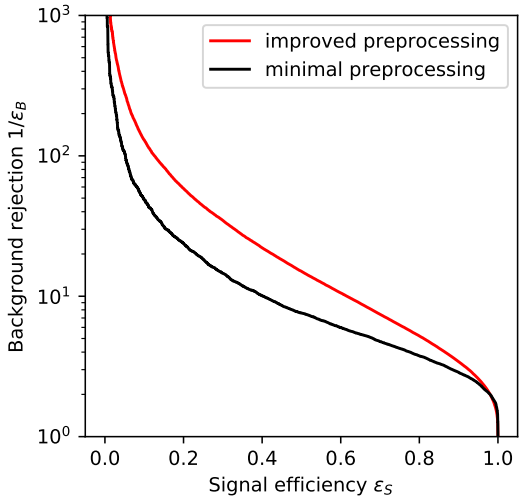


Figure 4.8: ROC curves for the image autoencoder classification result for improved and minimal preprocessing.

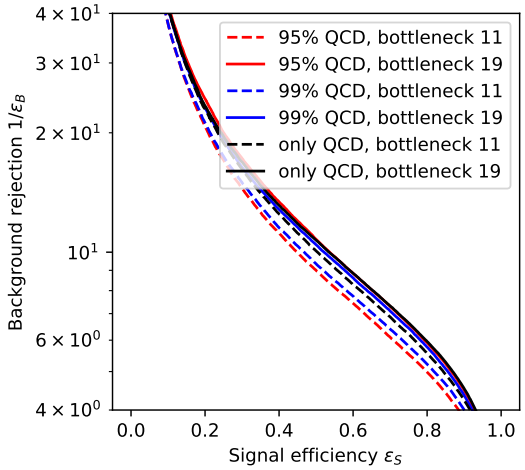


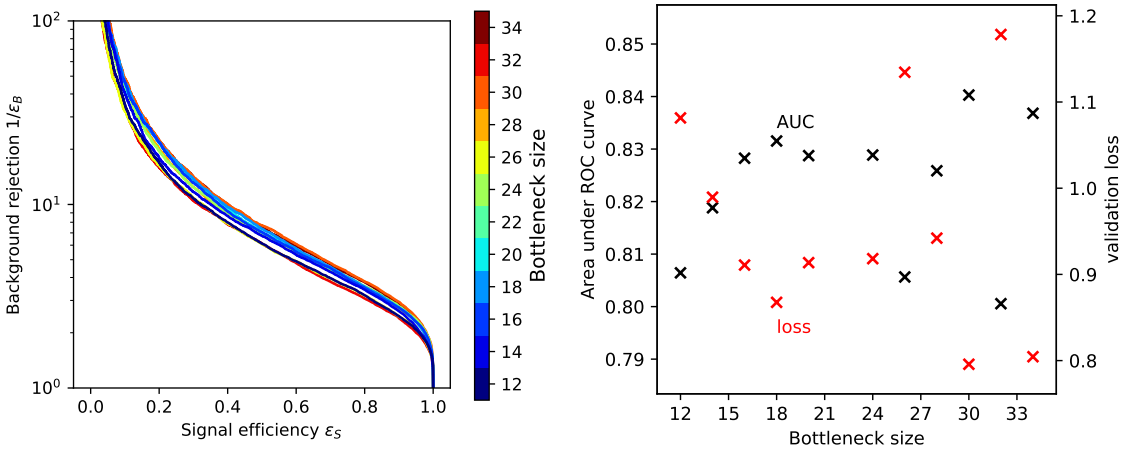
Figure 4.9: ROC curves for training on impure samples with different mixing ratios for the bottleneck sizes 11 and 19

The area under curve can even go up to 0.89 for good choices of the bottleneck size as we will show in the next section.

4.5.2 Testing bottleneck size

In this section, we will evaluate the effect of the bottleneck size on the loss and the discrimination power. We will again do this for both minimal and improved preprocessing to see the effects of the preprocessing, especially on the training stability. We expect the loss to be decreasing with increasing bottleneck size, because the network should be able to reproduce its input the better the less it has to compress the information of the jet images. Furthermore, we expect the AUC to increase with the bottleneck size at small bottlenecks, because the network should learn more general features of the jet images that are similar for both background and signal at very small bottlenecks.

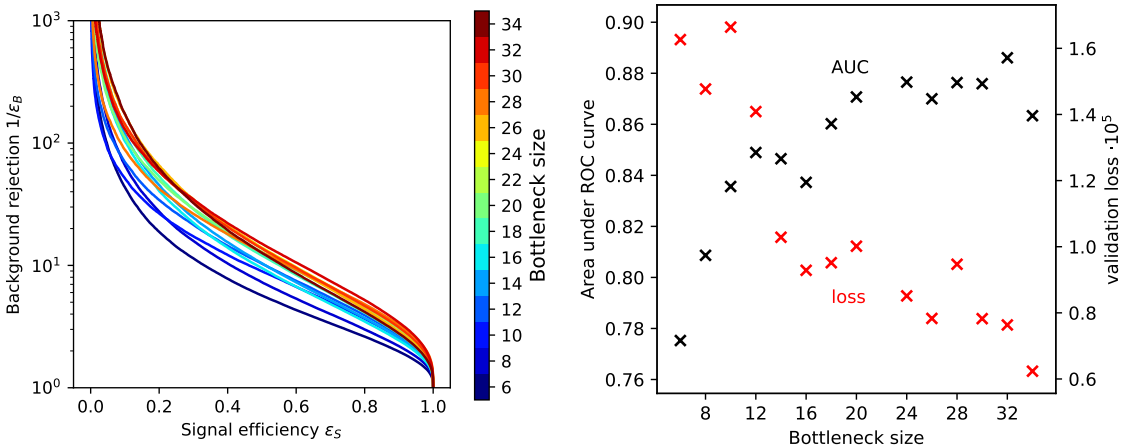
To test this, we again used the model architecture and training configuration from section 4.4.2. We only varied the number of units of the bottleneck layer while keeping all other model parameters the same. Then we created plots with the ROC curves for all different bottleneck sizes and a plot showing the validation loss and AUC to see if they match our predictions. The results for minimal preprocessing can be seen in Figure 4.10, and Figure 4.11 shows the results for the improved preprocessing. While for the improved preprocessing, the expected behavior is clearly visible with only small fluctuations, there are very large instabilities of the training on samples with minimal preprocessing, especially for large bottlenecks. The bottleneck size for the improved preprocessing should be at least 20 because then the AUC plateaus at approximately 0.88.



(a) ROC curves

(b) AUC and loss against bottleneck size

Figure 4.10: Discrimination results for the image autoencoder with minimal preprocessing for bottleneck sizes between 12 and 34.



(a) ROC curves

(b) AUC and loss against bottleneck size

Figure 4.11: Discrimination results for the image autoencoder with improved preprocessing for bottleneck sizes between 6 and 34.

4.5.3 Training on an impure sample

If we want to use the autoencoder on real data and especially on regions of phase space containing signal, we do not have a true classification whether an event is signal or background to compare against. However, most of the times, the dominant processes in these regions will be background events. To check if we can take advantage of that, we trained the autoencoder on two mixed samples, one with 5% tops and one with 1% tops. Other than that, the setup did not change. Figure 4.9 shows the ROC curves for two different bottleneck sizes. It can be seen that for larger bottlenecks, there is almost no difference in classification performance and the drop in performance is small for fewer bottleneck units too. Hence, the autoencoder can be used on an impure sample as long as the number of signal events is sufficiently small.

5 Autoencoder for jet constituents

So far, only autoencoder models working with jet images were examined. But as we have seen, the performance of the image autoencoder heavily depends on the image generation process. That raises the question whether better performance could be achieved by training an autoencoder directly with the calorimeter output without having to do image generation or preprocessing before.

Each event is now represented by up to 200 four-momentum vectors of the jet contituents. The vectors are sorted by their p_T. Very few jets have more than 120 constituents. The distributions for signal and background events are shown in Figure 4.3a.

5.1 Network architecture

5.1.1 Finding the optimal network architecture

The model architecture for constituents is very different from the architecture for images. Because they are not two-dimensional, we use a fully connected network instead of a convolutional network, so the network only contains dense layers and far fewer parameters have to be tuned. As most of the input four-vectors vanish most of the time or are not important for the signal-versus-background discrimination, the number of four-vectors given to the network can be reduced. We found that at least 40 constituents should be used to get good results. This is approximately equal to the position of the number-of-constituents distribution peak for QCD events.

So the main parameters to tune in the fully connected network are the number of dense layers, the number of units for each dense layer and their activation function. Unlike the image autoencoder, the fully connected autoencoder networks have to be sufficiently deep to provide good results. Nine dense layers proved to be a good compromise between classification performance and execution speed. The progression of the number of dense units was chosen to fall exponentially in the encoder part towards the bottleneck and then increase exponentially in the decoder part of the network.

Because the four-momenta can have negative components, the linear activation is best-suited for the output layer, because other activation functions that can have negative output values either have a limited output range like tanh or are not antisymmetric around zero like PReLU. Like for the image autoencoder, using PReLU instead of ReLU as activation function for all layers except for the last one strongly increased the classification performance.

In contrast to the image autoencoder, the training stability and the classification results improve for higher number of samples, so the autoencoder should be trained on the full training and testing sample for the image autoencoder. The loss converges faster for more

samples and the training thus takes fewer epochs, so the total training time does not increase linearly with the number of training samples.

5.1.2 LorentzLayer

For the DeepTopLoLa [4] tagger, two physics-inspired layers were introduced: the Lorentz Layer (LoLa) and the Combination Layer (CoLa). CoLa outputs the n input four vectors unchanged, but adds $m - n$ linear combinations of the four vectors, so all output vectors of the layer are (Lorentz invariant) four-vectors again. LoLa calculates the invariant mass and the transverse momentum for each constituent and linear combinations of the energies and distances between the four-vectors. With two trainable weight matrices $w^{(1)}$, $w^{(2)}$ and $\tilde{d}^2_{jm} = \eta^{\mu\nu}(\tilde{k}_j - \tilde{k}_m)_\mu(\tilde{k}_j - \tilde{k}_m)_\nu$, the use of CoLa and LoLa in DeepTopLoLa was as follows:

$$C = \begin{pmatrix} 1 & 0 & \dots & 0 & C_{1,n+1} & \dots & C_{1,m} \\ 0 & 1 & & \vdots & C_{2,n+1} & \dots & C_{2,m} \\ \vdots & \vdots & \ddots & 0 & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & C_{n,n+1} & \dots & C_{n,m} \end{pmatrix} \tag{5.1}$$

$$\tilde{k}_i = C_{ij} k_j \tag{5.2}$$

$$\hat{k}_j = \begin{pmatrix} m^2(\tilde{k}_j) \\ p_T(\tilde{k}_j) \\ w^{(1)}_{jm} E(\tilde{k}_m) \\ w^{(2)}_{jm} \tilde{d}^2_{jm} \end{pmatrix} \tag{5.3}$$

CoLa and LoLa were followed by a Flatten layer and a fully connected network. Using the two layers, a significant increase in tagging performance was achieved. We wanted a similar improvement for the autoencoder. However, as LoLa calculates the constituent masses and p_T while dropping the components of the four-momenta, it cannot be easily inverted which would be needed to use these layers in an autoencoder. We evaluated a minimal LoLa where the four-momenta were passed through unchanged and additional fields were added but this did not significantly improve the classification performance.

5.1.3 Boosted constituents

An alternative approach for a physics-inspired improvement of the network is to boost the jet constituents into the jet frame by applying a Lorentz transformation with the jet's three-velocity. Using this preprocessing, degrees of freedom containing no relevant information are removed from the input data in a physically motivated way. Again, we wanted to help the autoencoder by adding a CoLa and a minimal LoLa that added the invariant mass to the vectors. Indeed, for boosted constituents, the classification performance improved.

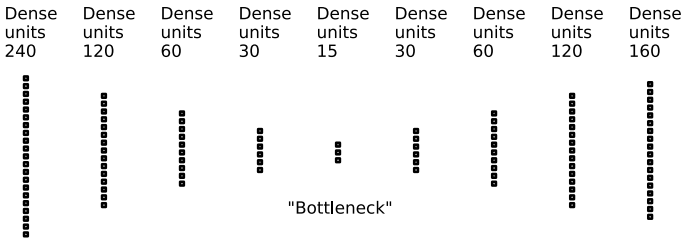


Figure 5.1: Final architecture of the fully connected autoencoder for the bottleneck size 15.

5.1.4 Final network architecture for constituents

Again we use KERAS with a THEANO backend to implement the training. Following the considerations from the last section, the following model yields good results for 40 constituents:

Flatten $\rightarrow$ *Dense(240)* $\rightarrow$ *Dense(120)* $\rightarrow$ *Dense(60)* $\rightarrow$ *Dense(30)* $\rightarrow$ *Dense(15)* $\rightarrow$ *Dense(30)* $\rightarrow$ *Dense(60)* $\rightarrow$ *Dense(120)* $\rightarrow$ *Dense(160)* $\rightarrow$ *Reshape*

PReLU activation is used for all layers except for the last one, where the activation is linear. Again, the network is trained up to 200 epochs with earlystopping after 10 epochs and the batch size is set to 128. The ADAM optimizer with a learning rate of 0.001 is used. The full Top Tagging Reference Sample is used for training, testing and validation. For other bottleneck sizes, only the number of dense units of the central layer (15 for the model above) is changed. For higher numbers of constituents, the following equation can be used to get an exponential progression of the number of units as for 40 constituents while keeping the bottleneck size constant. n is the number of constituents, u_i is number of units for i-th dense layer with $1 \leq i \leq 9$.

$$f_i = (16, 8, 4, 2, 1, 2, 4, 8) \tag{5.4}$$

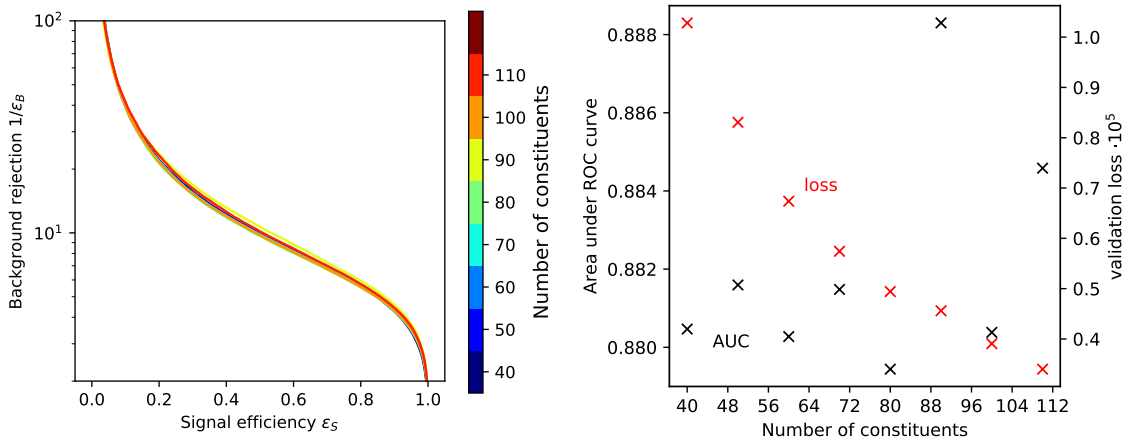
$$u_i = \begin{cases} \text{round}(15 * f_i^{\log(\frac{16}{40}n)/log(16)}) & \text{for } 1 \leq i \leq 8 \\ 4n & \text{for } i = 9 \end{cases} \tag{5.5}$$

5.1.5 Final network architecture for boosted constituents

Like for the non-boosted constituents, we used the 40 hardest constituents in the observer frame to train the network. We did not do any reordering according to the p_T of the constituents in the jet frame. We used the following progression of layers:

CoLa $\rightarrow$ *MinimalLoLa* $\rightarrow$ *Flatten* $\rightarrow$ *Dense(255)* $\rightarrow$ *Dense(160)* $\rightarrow$ *Dense(80)* $\rightarrow$ *Dense(40)* $\rightarrow$ *Dense(20)* $\rightarrow$ *Dense(10)* $\rightarrow$ *Dense(20)* $\rightarrow$ *Dense(40)* $\rightarrow$ *Dense(80)* $\rightarrow$ *Dense(160)* $\rightarrow$ *Reshape*

CoLa is configured as described above with $n = 40$ and $m = n + 10 + 1 = 51$ with the small addition that one four-vector containing the sum of all input four-vectors is appended. The minimal LoLa is defined as



(a) ROC curves

(b) AUC and loss against number of constituents

Figure 5.2: Discrimination results for the constituents autoencoder with no preprocessing for numbers of constituents between 40 and 120.

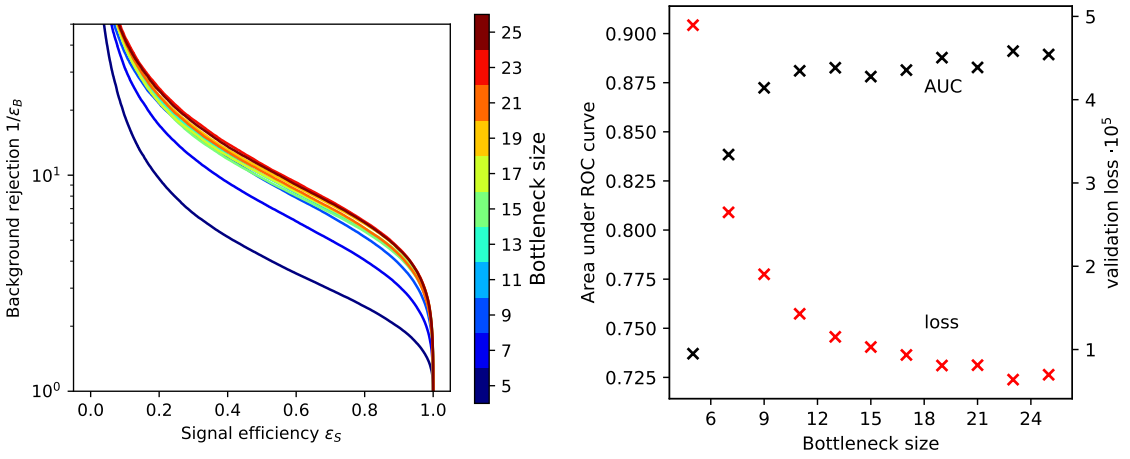
$$\hat{k}_j = \begin{pmatrix} \tilde{k}_j^0 \\ \tilde{k}_j^1 \\ \tilde{k}_j^2 \\ \tilde{k}_j^3 \\ w_{ji} m^2(\tilde{k}_i) \end{pmatrix} \tag{5.6}$$

with a trainable weight matrix w and the invariant mass $m(\tilde{k}_i)$ of the i-th input four-vector. PReLU activation is used for all the dense layers except for the last one, where the activation is linear. Again, the Adam optimizer with a learning rate of 0.001 is used and the batch size is 128. To get good convergence, training for 50 epochs with early-stopping after 5 epochs is sufficient. Similar to the non-preprocessed constituents, the bottleneck size can be changed by setting the number of units of the central dense layer (10 units in the layer progression given above).

5.2 Tests and results

5.2.1 Testing the number of constituents

First we want to test the influence of the number of constituents on the classification performance. To preserve the exponential layer progression, the numbers of dense units are chosen as described in equation 5.4. The results for the ROC curves and the losses and AUCs depending on the numbers of constituents are shown in Figure 5.2. It can be seen that – apart from small fluctuations – there is no difference in performance between lower and higher numbers of constituents, so 40 is a large enough number of constituents to get stable training with good performance. Thus, it will be used in all the following studies.



(a) ROC curves

(b) AUC and loss against bottleneck size

Figure 5.3: Discrimination results for the constituents autoencoder with no preprocessing for bottleneck sizes between 5 and 25.

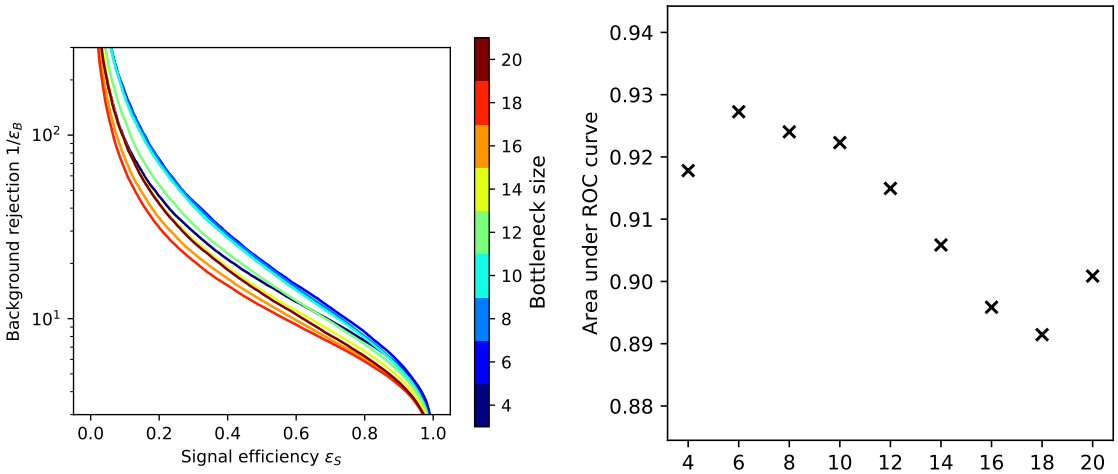
5.2.2 Testing bottleneck size

As for the image autoencoder, we want to test the effect of the bottleneck size, both for the boosted and non-preprocessed constituents. We used the training configurations given in the two previous sections and only changed the number of units of the bottleneck layer. The results for the ROC curves and the AUC depending on the bottleneck size for the jet constituents with no preprocessing are shown in Figure 5.3. Similar plots for the boosted constituents can be seen in Figure 5.4.

The results for the non-boosted constituents are qualitatively similar to those we saw for the image autoencoder: The discrimination power breaks down for very small bottlenecks and reaches a plateau at the bottleneck size 11. It can be seen that there were less fluctuations of the AUC compared to the autoencoder for images. However, for the boosted constituents, we see a completely different behavior. Instead of forming a plateau, the AUC peaks at small bottleneck sizes and then starts to decay.

5.2.3 Comparison of boosted and non-boosted jet constituents

As seen in Figures 5.3b and 5.4b, there is not only a qualitative difference in the dependency of the performance on the bottleneck but also a quantitative difference in the AUC scores that can be achieved: The best result for non-preprocessed constituents was 0.89 whereas the AUCs for boosted constituents went up to 0.93. As the AUC is only a one-dimensional quantity, we need to compare the ROC curves itself. Figure 5.5 shows the ROC curves of the best results for boosted and non-boosted constituents and also for the best image autoencoder as a reference. It can be seen that the autoencoder for images and non-preprocessed constituents perform similar in terms of the AUC, but in the signal efficiency region that is interesting for practical tagging purposes, the images autoencoder performs better. The performance of the autoencoder with boosted constituents is the best over the whole signal efficiency range.



(a) ROC curves

(b) AUC against bottleneck size

Figure 5.4: Discrimination results for the constituents autoencoder with boosted constituents for bottleneck sizes between 4 and 20.

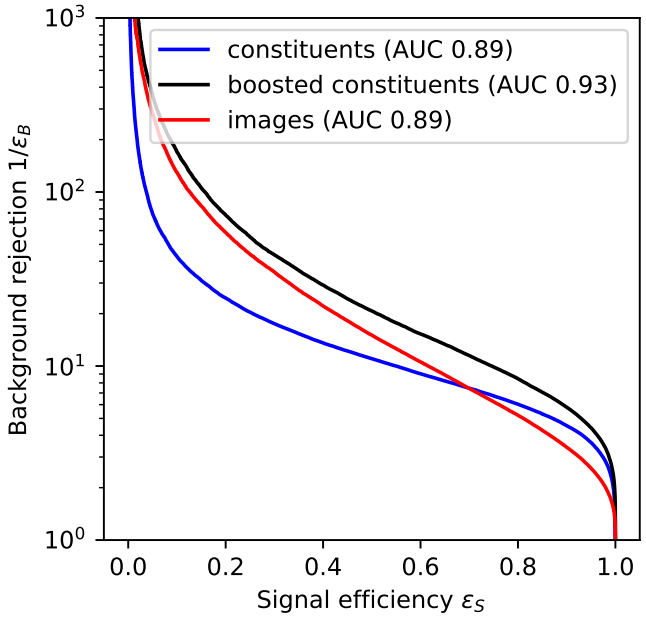


Figure 5.5: ROC curves of the best results for the image autoencoder and the jet constituent autoencoder (both boosted and non-preprocessed samples).

6 Taking out the mass with adversarial networks

6.1 Effect of the jet mass on the autoencoder result

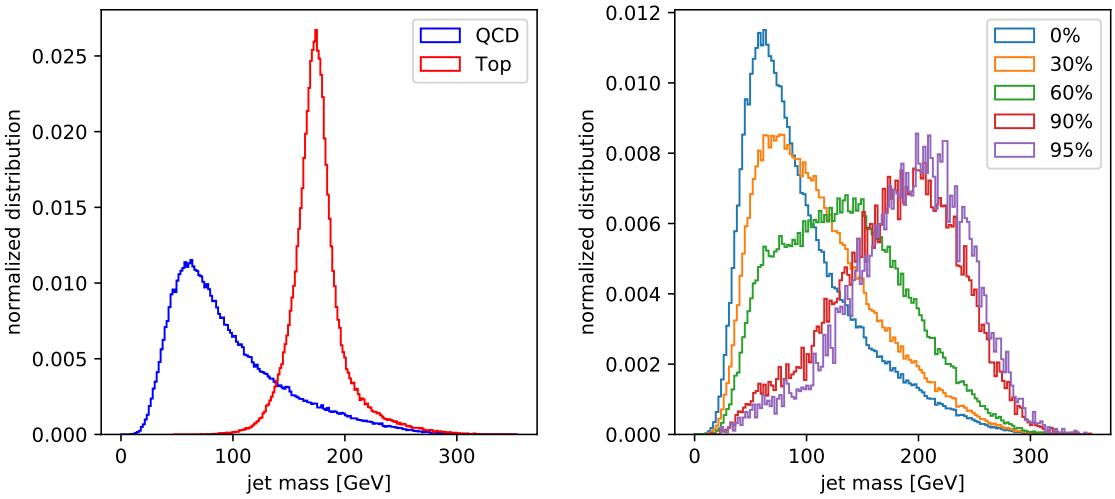
One of the most obvious observables to classify top and QCD events is the jet mass because there is a large difference between the mass distributions for top and QCD events (Figure 6.1a). Indeed, the autoencoder picks up the mass as one of the main criteria to classify events. This can be shown by plotting the mass distribution for QCD events that were erroneously classified as tops for different loss cutoffs. As seen in Figure 6.1b, a peak in the mass spectrum becomes visible for high cutoffs where most of the QCD events are excluded. If the autoencoder is focused on the mass, it is much harder for it to identify events that have a mass similar to QCD events. In this chapter, we describe an autoencoder that does not focus on the mass. Hence, it is able to learn other criteria that enable it to go beyond the simple scenarios where a good performance can be achieved by only looking at the mass.

6.2 Adversarial networks

Adversarial networks are a combination of two different networks working against each other, where the second part of the network is evaluating the output of the first part. The most common application of this type of network are Generative Adversarial Networks (GANs) where the first part generates outputs that are supposed to be similar to samples from a given dataset while the second part tries to tell if given samples were synthetically generated or taken from the dataset.

However, we want to use adversarial networks to impose the additional condition on the autoencoder that it should not learn to use the mass as a criterion for classification. This has been done for fully supervised tagging in Ref. [18]. The first part ("autoencoder part") of the network is simply the image autoencoder as described above. The second part ("adversarial part") of the network is a fully connected network that tries to extract the jet mass from the squared differences between autoencoder input and ouput, so the adversarial part must have N inputs and 1 output, where N is the number of pixels. With the squared differences $d_i = ((p_T^{norm,in})_i - (p_T^{norm,out})_i)^2$ between autoencoder input $p_T^{norm,in}$ and output $p_T^{norm,out}$ where i is an integer between 1 and N, we can define the loss of the autoencoder part as

$$L_{auto} = \frac{1}{N} \sum_{i=1}^{N} d_i \tag{6.1}$$



(a) Mass distribution for Tops and QCD

(b) QCD mass distributions with cutoff

Figure 6.1: (a) The signal and background mass distribution, calculated from the Top Tagging Reference Dataset validation sample. (b) The mass distribution of a pure QCD sample at different autoencoder loss cutoffs. Higher percentages mean that more events were removed from the distribution.

With ϕ being the trained adversarial network and M being the jet mass (which must be given), the loss of the adversarial part is defined as

$$L_{adv} = (\phi(d_1, \ldots, d_N) - M)^2 \tag{6.2}$$

The total loss is defined by subtracting these two losses, but with L_{adv} weighted by the Lagrange multiplier λ:

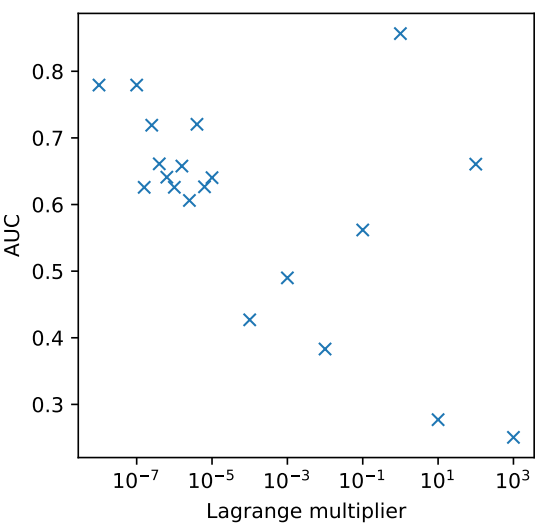
$$L_{tot} = L_{auto} - \lambda L_{adv} \tag{6.3}$$

The training for each batch is now done in two steps: First, the weights of the adversarial part are fixed and the network is trained using L_{tot} as objective, i.e. the total loss is minimized. Then, the weights of the autoencoder part are fixed and the network is trained using L_{adv} as objective. This way, the adversarial part learns to extract the jet mass as expected and the autoencoder tries to minimize the L_{auto} loss while trying to minimize the dependency of the loss on the jet mass to maximize L_{adv}. The balance between the two contributions can be set with the Lagrange multiplier. Adversarial training for jet constituents can be implemented analogous.

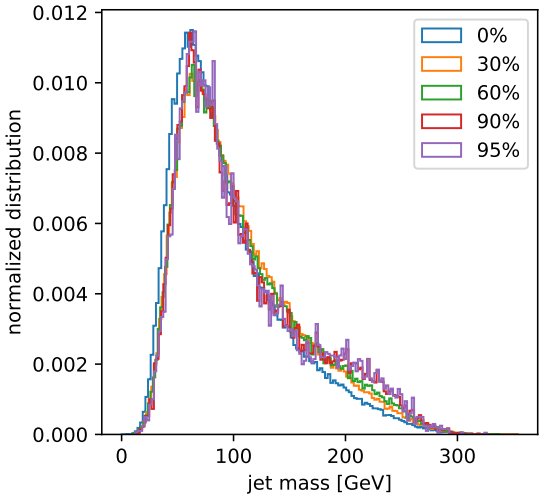
6.3 Training configuration and results

6.3.1 Jet images

We used the version of the final image autoencoder model for improved preprocessing as described in section 4.4.2 as the autoencoder part of the network. The architecture of the adversarial network was a deep fully-connected network:



(a) AUC versus Lagrange multiplier



(b) Pure QCD mass distributions for different cutoffs

Figure 6.2: (a) The AUC depending on the Lagrange multiplier λ at a learning rate of 10^{-4}. (b) The mass distribution of a pure QCD sample at different autoencoder loss cutoffs. The adversarial autoencoder was trained with $\lambda = 3.89 \cdot 10^{-6}$ and learning rate 10^{-4}. Higher percentages mean that more events were removed from the distribution.

Input(1600) $\rightarrow$ *Dense(800)* $\rightarrow$ *Dense(400)* $\rightarrow$ *Dense(200)* $\rightarrow$ *Dense(100)* $\rightarrow$ *Dense(50)* $\rightarrow$ *Dense(25)* $\rightarrow$ *Dense(10)* $\rightarrow$ *Dense(1)*

The network was trained on a subset (100,000 training samples and 25,000 testing samples) of the image samples with improved preprocessing. Each training epoch is split up into batches with a batch size of 128. For each batch, as described above, the autoencoder part of the network is trained first and afterwards, the adversarial part is trained. It is important to notice that the learning rate of the optimizer has to be much lower than for a regular autoencoder, otherwise it is hard for the two parts of the network to adapt to each others learning progress. To get useful results, it is now necessary to find an adequate value for the Lagrange multiplier λ. For very small λ, the network behaves similarly to the regular autoencoder because the contribution of the adversarial loss to the total loss that is used as an objective for the autoencoder training is negligible. On the other hand, for very large λ, the total loss is dominated by the adversarial loss and the autoencoder cannot be trained properly. In Figure 6.2a, the AUCs for a scan over multiple orders of magnitude of λ are shown. For $\lambda \geq 10^{-4}$, there are large fluctuations of the AUC and often, it even is below 0.5. By checking the output images of the autoencoder, we observed that the autoencoder only learns to always output the same image in these cases. The interesting region where both loss contributions have similar sizes was $10^{-7} \leq \lambda \leq 10^{-5}$. For $\lambda = 3.89 \cdot 10^{-6}$ at a learning rate 10^{-4}, we had a result with an AUC of 0.74 where indeed the training did not learn to use the mass for discrimination as seen in Figure 6.2b as the mass distribution for a pure QCD sample has almost the same shape for different loss cutoff values.

However, as seen in Figure 6.2a, the network has stability problems in the interesting region, so instead of the desired minimal shaping of the mass, we often observed a bifurcated mass distribution with one peak at the same position as the peak in the QCD mass

distribution and a second peak at higher masses. The reason for these instabilities is that it is hard for the adversarial part of the network to keep up with the autoencoder during training. As a simple improvement, we repeated the training of the adversarial part multiple times for every batch. Indeed, the stability was much better and we were able to get reproducible training results with a mass shaping behaviour similar to our first successful result, if we chose 8 or 10 as number of repetitions. Other methods like pre-training the adversarial part of the network for some epochs before the training of both parts of the network started or training only the adversarial part every second or third epoch were not successful and yielded bifurcated mass distributions again. Also, for learning rates above 10^{-4}, the suppression of the mass became worse.

So far, we used a MSE loss function for the adversarial part and the single output value of the network was compared to the mass directly. However, as seen above, this makes local minima possible, where the network learns to output the mean jet mass instead of the jet mass of the input events, which could explain the mass distribution bifurcation issue. Indeed, in Ref. [18], it was proposed to use an adversarial network with 10 outputs instead, where each output stands for one decile of the mass distribution (i.e. one histogram bin containing 10% of the events). Categorical crossentropy was used as loss function. We modified the adversarial part to have the following architecture:

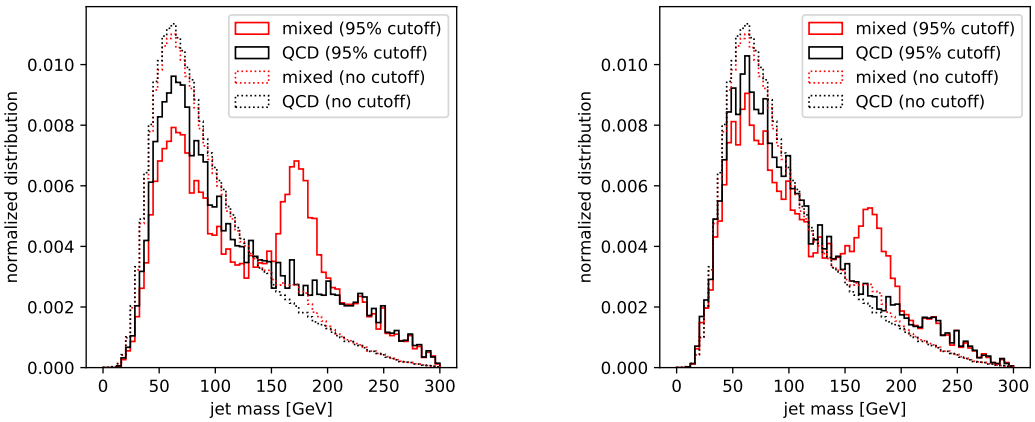
Input(1600) $\rightarrow$ *Dense(800)* $\rightarrow$ *Dense(400)* $\rightarrow$ *Dense(200)* $\rightarrow$ *Dense(100)* $\rightarrow$ *Dense(50)* $\rightarrow$ *Dense(25)* $\rightarrow$ *Dense(10)* $\rightarrow$ *Dense(12)*

All layers except for the last one have ReLU activation, the last layer has SoftMax activation because the normalization is needed for the categorical crossentropy to work properly. The output layer has 12 nodes instead of 10 because the bins are calculated for a pure QCD sample, but in the final testing and also when training with a mixed sample, samples with higher or lower jet masses can occur. The results were indeed more stable than for the previous version and less epochs were needed to achieve similar or better decorrelation of the mass. Again, 10^{-4} proved to be a good choice for the learning rate. Because of the changed loss function for the adversarial part, the Lagrange multipliers changed. We had good results in terms of mass decorrelation for $\lambda = 5 \cdot 10^{-4}$. For the adversarial autoencoder with MSE loss, 180 epochs were not enough for full convergence in some cases. In contrast, with CCE loss, even only 20 epochs were often enough to achieve good mass shaping suppression. The average AUC for these settings was 0.70.

Finally, we want to show how the amount of mass decorrelation can be controlled with the Lagrange multiplier. To visualize the mass decorrelation, we plot four mass distributibutions for each result:

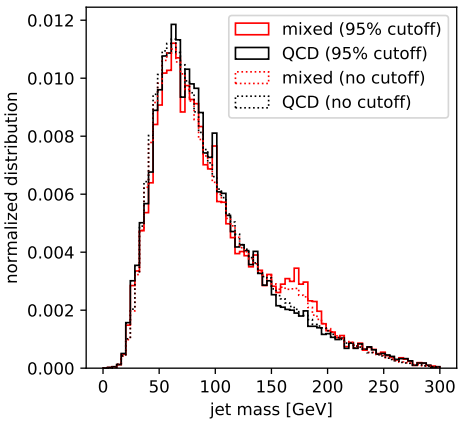
- a pure QCD sample,
- a mixed sample with 97% QCD,
- the pure QCD sample with 95% of the events with the lowest autoencoder losses removed and
- the mixed sample with 95% of the events with the lowest autoencoder losses removed.

Figure 6.3 shows results for three different Lagrange multipliers. The learning rate was 10^{-4} and the network was trained for 40 epochs. For $\lambda = 1 \cdot 10^{-4}$, the resulting AUC is 0.78, for $\lambda = 5 \cdot 10^{-4}$, it is 0.70 and for $\lambda = 1 \cdot 10^{-3}$, it is 0.62. We can see in the mass

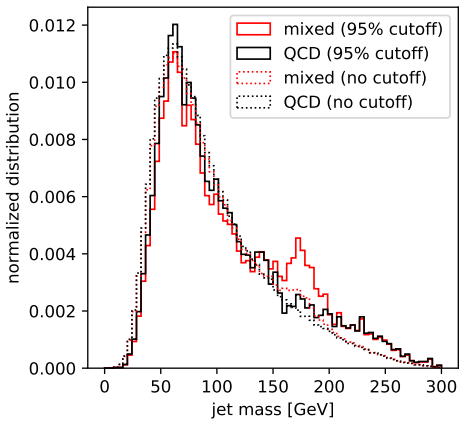


(a) $\lambda = 1 \cdot 10^{-4}$

(b) $\lambda = 5 \cdot 10^{-4}$



(c) $\lambda = 1 \cdot 10^{-3}$



(d) Impure training sample with 3% tops.

Figure 6.3: Mass distributions for pure QCD and a mixed sample with 97% QCD and the corresponding distributions when 95% of the events with lowest autoencoder losses are removed. (a - c) Results for a pure training sample at three different Lagrange multipliers. (d) Result for a mixed training sample.

distribution plots, that the autoencoder can be tuned towards a better mass decorrelation by using higher λ with the tradeoff that the discrimination power decreases.

6.3.2 Training on an impure sample

Like for the non-adversarial autoencoder, we also evaluated the performance for training on an impure sample. We used training and testing samples with 97% QCD events and 3% top events. Again, the parameters we had to tune were the Lagrange multiplier λ, the learning rate and the bottleneck size. We had good results in terms of mass shaping while still having a high enough discrimination (AUC 0.69) power for $\lambda = 3 \cdot 10^{-4}$, a learning rate of 10^{-4}, a bottleneck size of 10 and training for 40 epochs. Figure 6.3d shows the resulting mass distributions. That shows that even for the adversarial training, the autoencoder can be successfully trained on an impure sample.

7 Using the autoencoder to find new physics

7.1 New physics samples

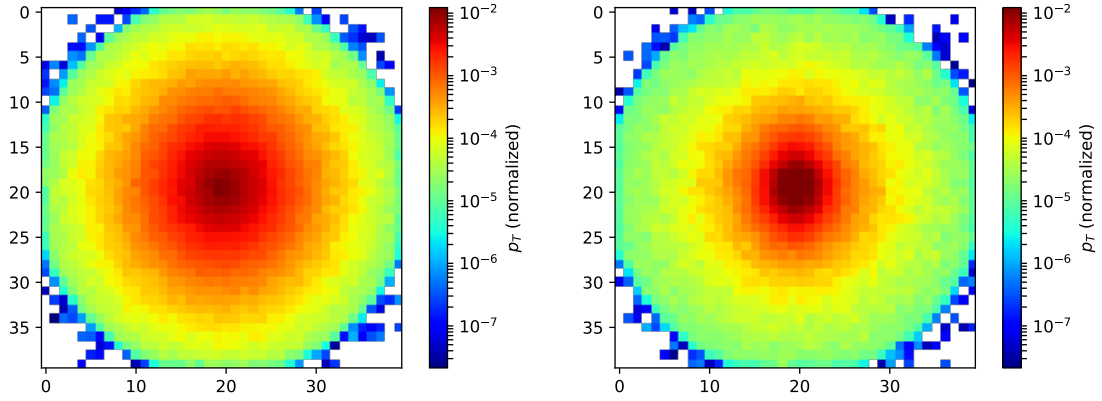
So far, we only tested the autoencoder on top tagging samples as a benchmark. In this chapter, we want to use other samples to see if the autoencoder would be able to identify physics beyond the Standard Model when trained on a QCD background. Using a similar tool chain as for the Top Tagging Reference Sample, we generated new samples. For each of the new physics scenarios described in the following, the QCD background of the corresponding region of phase space was generated.

7.1.1 Heavy Higgs

We considered an extended Higgs sector with a heavy scalar Higgs boson that has the same mass as the top quark. The heavy Higgs decays to two pseudoscalars which decay into two charm quarks each. We chose the mass of the pseudoscalars to be approximately two times the mass of the charm quark. We prepared two pure QCD background samples for the training and one mixed sample to test the classification. We then generated the jet images again. The preprocessing methods used for the top tagging sample are specifically aimed at decays into three particles, as the images are shifted, rotated and flipped to cope with the first, second and third prong. However, as that is no longer the case for the new samples, we used minimal improved preprocessing: We centered the centroid of the (η, ϕ) plane weighted by p_T and then pixelated the images. The resolution was 40×40 again and we used the range $[-0.75, 0.75]$ for both η and ϕ. Figure 7.1 shows the resulting images.

7.1.2 Dark Showers

The next type of new physics samples we want to consider are Dark Showers, where we assume a strongly coupled dark sector. A dark quark-antiquark pair is produced and hadronizes to dark mesons which in turn decay back to visible matter. The mass of the dark quarks and the dark mesons can be adjusted in order to have different scenarios to test the autoencoder. While having similar experimental signatures to the QCD background, the jet substructure is different. That makes it an interesting test for the autoencoder. The samples were generated in Pythia with the hidden valley model implemented there. We generated samples with quark masses of 50 GeV and 100 GeV and mesons with mass 10 GeV and samples with 200 GeV quark mass and the meson masses 10,20,40,80 and 100 GeV. There is only one QCD training and testing sample that is used as background for all the signal samples. As the jet mass is approximately the quark mass, especially the sample with 50 GeV is interesting because the QCD jet mass peaks at a similar point. Again, the images were generated using minimal improved preprocessing. As for the Heavy Higgs samples, we used a resolution of 40×40 and the range $[-0.75, 0.75]$ for η and ϕ.



(a) Signal events

(b) Background events

Figure 7.1: Mean jet images for heavy Higgs signal and QCD background events with image size 40×40 and mean improved applied. (Means taken over the full validation sample.)

7.2 Results

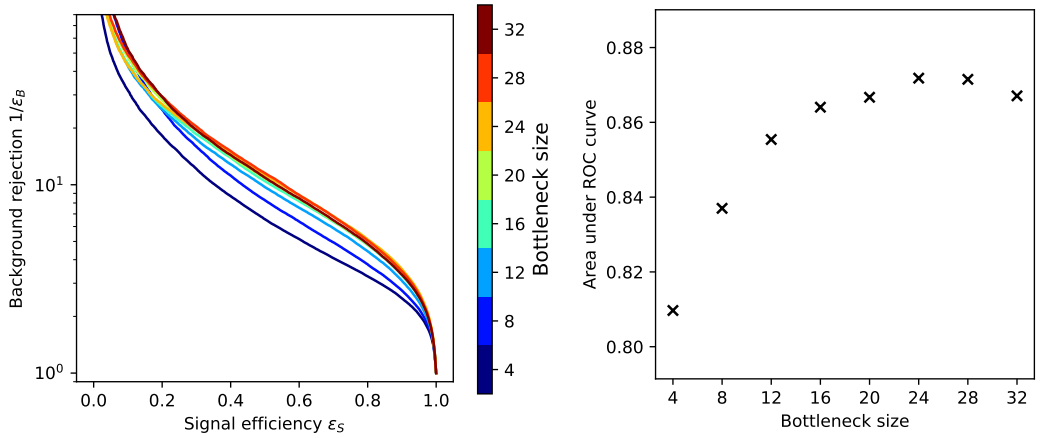
7.2.1 Heavy Higgs

Like for the top tagging, we evaluated the performance of the autoencoder (without adversarial training) for Heavy Higgs events at different bottleneck sizes between 4 and 32. The training setup and network architecture was again like in section 4.4.2. The results for the ROC curves and the AUC depending on the bottleneck size are shown in Figure 7.2. It can be seen that the curve in Figure 7.2b has a shape similar to the one for the top tagging sample. Again, the performance starts to plateau at a bottleneck size of 20. With an AUC of 0.87, the best performance is not as good as for the top tagging, although the signal events have similar masses. This could be because of the process-specific preprocessing that was used for the autoencoder for tops, so the neural network has a small advantage when classifying fully preprocessed top events. However, it also shows that the autoencoder still has good discrimination power for a different physics scenario than the one it was initially optimized for.

7.2.2 Dark Showers

The classfication power of the autoencoder for Dark Showers was again evaluated by training autoencoders with bottleneck sizes between 4 and 32. As there is only one QCD background training sample, the network was only trained once for every bottleneck size. Then, ROC curves for all the different signals were computed. Figure 7.3 shows the results for the AUC scores.

It can be seen that the mass m_q of the dark quark has the largest impact on the classification performance. As the network is not mass-decorrelated, the most plausible explanation for that observation is that for $m_q = 100$ GeV and especially $m_q = 50$ GeV, the jet mass distribution is more similar to the QCD jet mass distribution, so the jet mass is not a suitable discrimination observable any more. Hence, the jet substructure becomes relevant to distinguish the Dark Showers from the background. Furthermore, it can be seen that for



(a) ROC curves

(b) AUC against bottleneck size

Figure 7.2: Discrimination results of the image autoencoder for Heavy Higgs samples for bottleneck sizes between 4 and 32.

higher meson masses, the AUC is slightly improved. However, this improvement is only visible when comparing the samples with different meson masses for the same training run, as the fluctuations between different training runs are much larger. This can be seen at the bottleneck sizes 20 and 24 in Figure 7.3. For $m_q = 50GeV$, no increase in performance for larger bottlenecks is visible. For larger quark masses, increasing AUCs are visible (apart from fluctuations), but unlike the results for top or Higgs tagging, there is no clearly visible plateau. The best AUC that was achieved for one of the Dark Shower variants was 0.80.

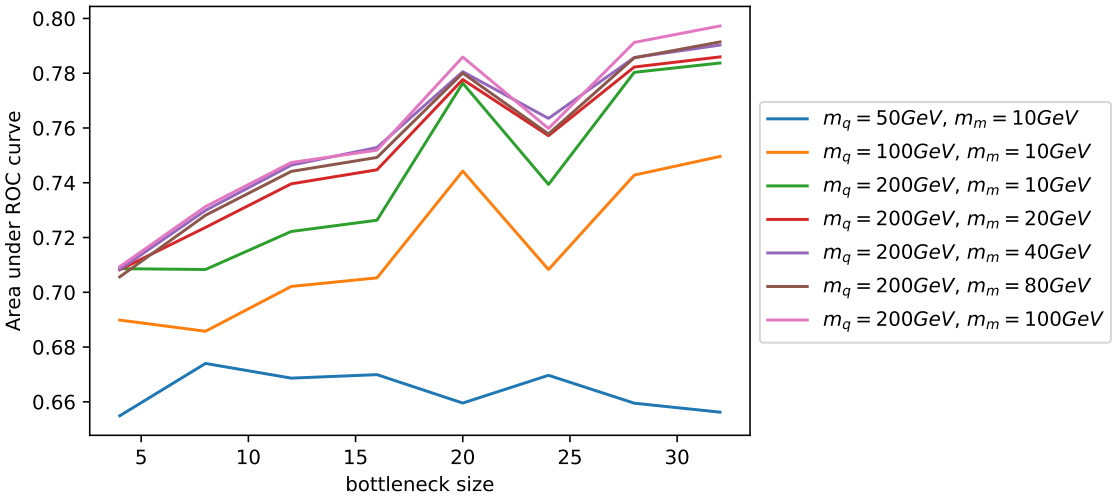


Figure 7.3: AUCs for all dark shower samples for the image autoencoder trained on QCD background with bottleneck sizes between 4 and 32. The connecting lines between the points were only added for better readability and do not correspond to actual classification results. m_q is the mass of the dark quark and m_m is the mass of the mesons.

8 Conclusion

In this thesis, we demonstrated that autoencoders can be used to tag signal events even though they were only trained on the QCD background. We used top tagging as a benchmark scenario to test our autoencoders. First, we implemented an autoencoder for jet images and found a model to optimize the discrimination performance. We examined two different preprocessing methods: a minimal preprocessing where the images were only centered at their hardest constitutent and an improved preprocessing specifically designed for the three-prong top decay. With minimal preprocessing, we showed that an AUC of 0.84 is possible, whereas for the improved preprocessing, the AUC can go up to 0.89. In both cases, an effect of the classification performance on the bottleneck was observed: For small bottleneck sizes, the AUC was small and, starting around a bottleneck size of 20, reached a plateau. Furthermore, we showed that with only a minor decrease in performance, the autoencoder can also be trained on impure samples.

Next, we implemented an autoencoder directly trained on the jet constituent four-vectors. We found that varying the numbers of constituents used for training only had a negligible effect on the training performance above 40 constituents, thus we used 40 constituents for the following tests. Then we examined a pre-processing method for the constituents where they were Lorentz-boosted into the jet frame. We again looked at the effect of the bottleneck size and saw a similar behavior for the non-preprocessed constituents as for the images, whereas for the boosted constituents, the discrimination power peaked at small bottlenecks. With non-boosted constituents, we achieved a performance of AUC 0.89. This is the same as the best AUC for the image autoencoder, however, the images performed better in the relevant signal efficiency regions. For boosted constituents, the best performance was AUC 0.93. Comparing the ROC curves showed that the latter variant of the autoencoder indeed performed better than both the image autoencoder and the one for unboosted constituents over the whole signal efficiency range.

Subsequently, we successfully decorrelated the autoencoder for images from the jet mass using adversarial networks. We compared mean squared error and categorical crossentropy as loss functions for the adversarial networks and found the latter to have better convergence during training. Using different Lagrange multipliers λ, we were able to continuously sweep between an autoencoder without any additional conditions and full mass decorrelation. We found that there is a tradeoff between discrimination performance in terms of the AUC and the mass decorrelation. $\lambda = 5 \cdot 10^{-4}$ (the absolute value depends on the model and the choice of sample) was a good compromise with an AUC of 0.70 and only a small dependency of the classification on the jet mass.

Finally, we looked at two new physics processes. We considered an extended Higgs sector with a Heavy Higgs that has the same mass as the top quark, and a strongly coupled dark sector with a dark quark hadronizing into dark mesons decaying back into visible particles. We trained the autoencoder for images on the corresponding QCD backgrounds and tested the dependency of the discrimination performance on the bottleneck size. While for the Heavy Higgs sample this dependency was similar to the corresponding curve for

top tagging with a maximal AUC of 0.87, the course of the bottleneck versus AUC curve was strongly dependent on the dark quark mass for the Dark Shower samples. This is because the heavy scalar has the same mass as the top mass, whereas for dark quarks with masses similar to the QCD jet mass peak, the network has to use the jet substructure for discrimination.

Let us formulate the key insights of this thesis:

- No matter if the autoencoder is based on jet images or on jet constituents, the preprocessing crucially affects the classification performance. For both types of input data, we saw a substantial increase in performance when we used preprocessing mechanisms inspired by the physical nature of the data.
- While the final version of the autoencoder for constituents clearly had a better performance for top tagging than the image based variant, both types of autoencoder are useful. Especially when we decorrelated the classification from the mass, the autoencoder for images had a higher training stability and the amount of mass decorrelation could be easily tuned.
- We have seen that both the simple and the mass-decorrelated version of the autoencoder can also be trained on an impure sample with only a small decrease in performance. That means that it would not even be necessary to train the autoencoder on a sample with known labels, as long as the ratio between background and signal events is known to be sufficiently large.
- The autoencoder is not only able to tackle top tagging, but because of it being based on anomaly detection rather than known labels, it can also be used for arbitrary classification problems in particle physics, even if we do not fully understand the processes behind the signal.

This thesis leaves open many avenues of future research. For example, the autoencoder could be tested on more physics scenarios. Also, there is still much room for improvements of the autoencoder architectures we developed: While much time and effort went into optimizing the model for jet images, further research is needed refine the model and tune the training parameters for jet constituents and especially the adversarial autoencoder.

An interesting topic for further research is the compressed representation of the events found by the autoencoder that can be extracted from the bottleneck. For instance, one could look for correlations between the bottleneck units and actual physical properties, for example jet variables commonly used for tagging. Also, instead of using the loss function as classification score, well-established anomaly detection methods that are suited for data with fewer dimension than the calorimeter output or jet images, like one-class support vector machines [19], isolation forests [20] or – more recent – one-class neural networks [21], could be applied to the compressed representation. Finally, the next step to further test the autoencoder would be to run it on experimental data instead of simulated events, as the studies with impure samples have shown that the training sample can contain a small fraction of signal events. Thus, both training and evaluation could be done on experimental data which opens a wide field of application for the autoencoder.

Bibliography

[1] Georges Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Phys. Lett.* B716 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[2] Serguei Chatrchyan et al. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Phys. Lett.* B716 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021. arXiv: 1207.7235 [hep-ex].

[3] Gregor Kasieczka et al. "Deep-learning Top Taggers or The End of QCD?" In: *JHEP* 05 (2017), p. 006. DOI: 10.1007/JHEP05(2017)006. arXiv: 1701.08784 [hep-ph].

[4] Anja Butter et al. "Deep-learned Top Tagging with a Lorentz Layer". In: (2017). arXiv: 1707.08966 [hep-ph].

[5] Gregor Kasieczka, Michael Russel, and Tilman Plehn. *Top Tagging Reference Dataset*. 2018. URL: https://docs.google.com/document/d/1Hcuc6LBxZNX16zjEGeq16DAzspkDC4nDTyjMp1b
edit.

[6] Chase Shimmin et al. "Decorrelated Jet Substructure Tagging using Adversarial Neural Networks". In: *Phys. Rev.* D96.7 (2017), p. 074034. DOI: 10.1103/PhysRevD.96.074034. arXiv: 1703.03507 [hep-ex].

[7] Wikimedia Commons. *Standard Model of Elementary Particles*. 2006. URL: https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg.

[8] Y. Fukuda et al. "Evidence for oscillation of atmospheric neutrinos". In: *Phys. Rev. Lett.* 81 (1998), pp. 1562–1567. DOI: 10.1103/PhysRevLett.81.1562. arXiv: hep-ex/9807003 [hep-ex].

[9] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. "The Anti-k(t) jet clustering algorithm". In: *JHEP* 04 (2008), p. 063. DOI: 10.1088/1126-6708/2008/04/063. arXiv: 0802.1189 [hep-ph].

[10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: (2010).

[11] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[12] Torbjörn Sjöstrand et al. "An Introduction to PYTHIA 8.2". In: *Comput. Phys. Commun.* 191 (2015), pp. 159–177. DOI: 10.1016/j.cpc.2015.01.024. arXiv: 1410.3012 [hep-ph].

[13] J. de Favereau et al. "DELPHES 3, A modular framework for fast simulation of a generic collider experiment". In: *JHEP* 02 (2014), p. 057. DOI: 10.1007/JHEP02(2014)057. arXiv: 1307.6346 [hep-ex].

[14] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. “FastJet User Manual”. In: *Eur. Phys. J.* C72 (2012), p. 1896. DOI: 10.1140/epjc/s10052-012-1896-2. arXiv: 1111.6097 [hep-ph].

[15] Sebastian Macaluso and David Shih. “Pulling Out All the Tops with Computer Vision and Deep Learning”. In: (2018). arXiv: 1803.00107 [hep-ph].

[16] François Chollet et al. *Keras*. https://keras.io. 2015.

[17] Rami Al-Rfou et al. “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[18] Chase Shimmin et al. “Decorrelated Jet Substructure Tagging using Adversarial Neural Networks”. In: *Phys. Rev.* D96.7 (2017), p. 074034. DOI: 10.1103/PhysRevD.96.074034. arXiv: 1703.03507 [hep-ex].

[19] Larry M. Manevitz and Malik Yousef. “One-class Svms for Document Classification”. In: *J. Mach. Learn. Res.* 2 (Mar. 2002), pp. 139–154. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944790.944808.

[20] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation-Based Anomaly Detection”. In: *ACM Trans. Knowl. Discov. Data* 6.1 (Mar. 2012), 3:1–3:39. ISSN: 1556-4681. DOI: 10.1145/2133360.2133363. URL: http://doi.acm.org/10.1145/2133360.2133363.

[21] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. “Anomaly Detection using One-Class Neural Networks”. In: *CoRR* abs/1802.06360 (2018). arXiv: 1802.06360. URL: http://arxiv.org/abs/1802.06360.

Acknowledgements

First, I would like to thank Tilman Plehn for giving me the opportunity to work on this interesting project and to gain first experiences in research. A special thanks goes to Jennifer Thompson for answering many questions and proof-reading this thesis. Furthermore, I would like to thank Gregor Kasieczka for his extremely helpful support throughout the project. Also, I want to thank Nicholas Kiefer and Michel Luchmann (who also implemented the improved preprocessing used in this thesis) for the interesting discussions about machine learning and physics. Finally, I would like to thank the whole LHC physics group for providing such a nice environment, and for the cake and ice cream.

Erklärung

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

Heidelberg, den 23. August 2018

Theo Heimel