Department of Physics and Astronomy
University of Heidelberg

Master Thesis in Physics
submitted by

Sascha Daniel Diefenbacher

born in Tuttlingen (Germany)

2019

Tagging Whole LHC Events with Capsule Networks

This Master Thesis has been carried out by Sascha Daniel Diefenbacher at the
Institute for Theoretical Physics in Heidelberg
under the supervision of
Prof. Tilman Plehn

Abstract

We employ capsule networks, a novel approach to image classification in machine learning, to tag whole LHC events. The capsule networks' ability to simultaneously learn sub-jet features and event level kinematics makes them a well suited tool for this task. Initially, we compare them to established convolutional neural networks using top vs QCD events. Further, we show the capsule networks' power by tagging a heavy $Z' \to t\bar{t}$ resonance against a QCD mediate di-top and light jet background, both individually and as a combined, mixed background. Here, we also show how the capsule structure lets us interpret the networks results more easily. Finally, we demonstrate the capsule networks ability to handle complex, high activity events, by tagging associated top-Higgs production.

Zusammenfassung

Zur Klassifizierung kompletter LHC Events verwenden wir sogenannte Kapselnetzwerke, welche einen neuen Ansatz im Bereich Machine Learning liefern. Die Fähigkeiten der Kapselnetzwerke, gleichzeitig sub-Jet Details und eventübergreifende, kinematische Eigenschaften zu erkennen, machen sie zu einem sehr gut geeigneten Werkzeug für diese Aufgabe. Zu Beginn vergleichen wir die Kapselnetzwerke mit etablierten Convolutional Neural Networks unter Zuhilfenahme von Top- und QCD-Jet Events. Des Weiteren demonstrieren wir die Stärke der Kapselnetzwerke indem wir Events aus dem Zerfall eines schweren Z''s in zwei Top-Jets vom QCD-Hintergrund trennen. Dabei betrachten wir auf der einen Seite einen Hintergrund aus zwei top-Jets, und auf der anderen Seite einen aus zwei leichten Jets. Anschließend testen wir das Netzwerk mit einer Kombination der beiden Hintergründe. An diesem Punkt zeigen wir auch, wie sich die Kapselnetzwerkergebnisse interpretieren lassen. Schlussendlich legen wir am Beispiel von assoziierter Higgs-Top Produktion dar, dass die Kapselnetzwerke auch in der Lage sind hoch komplexe Prozesse zu klassifizieren.

Contents

1 Introduction **1**

2 Jet Physics at the LHC **3**

2.1 The Large Hadron Collider 3

2.2 From Detector to Calorimeter Image 3

2.3 Jet Definitions 6

2.4 Top and QCD Jets 7

3 Neural Networks **9**

3.1 Neurons and Layers 9

3.1.1 Dense Layers 9

3.1.2 Pooling Layers 10

3.1.3 Convolutional Layers 11

3.2 Activation Functions 13

3.3 Loss Functions 14

3.4 Gradient Descent 16

3.5 ROC Curves and Classifiers 18

4 Capsule Network **23**

4.1 Motivation 23

4.2 Capsule Layers 26

4.2.1 Primary Caps 27

4.2.2 Routing Layers 28

4.3 Dynamic Routing 29

4.4 Squashing Functions 32

4.5 Margin Loss 34

4.6 ϕ-Padding 35

4.7 Choice of Classifier 37

5 Whole Event Inputs **39**

5.1 Event Generation 39

5.2 Data Storage 39

6 Top vs QCD Dataset **41**

7 Heavy Mediator Decay **45**

7.1 QCD Background 45

7.2 $t\bar{t}$ Background 48

7.3 Combined Backgrounds 51

7.4 Understanding Capsule Entries 54

8 $t\bar{t}H$ **59**

9 Conclusion and Outlook **65**

1 Introduction

When analyzing LHC events, the classical approach consists of choosing a set of observables specific to the process of interest and then using these observables for a cut based analysis. Over the past years, machine learning methods have started to revolutionize this process. Initially, in the form of Boosted Decision Trees [1], which outperformed standard cut-and-count approaches thanks to their increased flexibility, and more recently through the introduction neural networks. The area where this innovation arguably had the biggest impact is jet tagging. Here, the power of image-classification neural networks [2–4] was harnessed by taking a jet, and translating it into a calorimeter image in the η-ϕ plane. The most prominent type of these networks, convolutional neural networks, had originally been used outside of physics for applications such as facial recognition [5] or self driving-cars [6]. Such image based networks, alongside alternative architectures that use e.g. 4-vectors as input, are able to discriminate quark and gluon jets [7–12], as well as to tag Higgs- [13, 14] and top-jets [15–22]. However, all of these analysis methods have one similarity, they require someone to decide what part of an LHC event is most relevant. Either by deciding on which observables to cut on, or by choosing which jet to use for tagging. Therefore, what we aim to accomplish in this thesis is skipping this selection step. We want to classify whole LHC events using only the raw calorimeter and tracking information, to effectively have a purely data driven approach.

In the field of image classification, capsule networks are essentially an extension of standard convolutional neural networks. Capsules have seen previous application in astrophysics [23, 24], but had so far received little attention in particle physics. However, their ability to adapt to shifting feature positions in an image and handle both large and small features simultaneously makes them appropriate for full event analysis.

The results of this thesis are part of an already published paper [25]. Section 7.4 specifically is based on the results of Hermann Frosts bachelor thesis. In the thesis, we will first discuss the basics of LHC jet physics in section 2 and introduce the general principles of machine learning in section 3. Then, we will elaborate on the advantages of capsule networks and explain their underlying principles section 4, as well as describe how we handle full event data files in section 5. Subsequently, we apply different capsule architectures to a series of samples, initially testing them on the top vs QCD challenge data set [21, 26] in section 6. Following this, we investigate the capsule networks' ability to tag a heavy mediator resonance against both a QCD and $t\bar{t}$ background in section 7 and finally in section 8, we apply them to a complex, high activity event class in the form of associated top pair Higgs production. Finally, we summarize our results in section 9.

2 Jet Physics at the LHC

2.1 The Large Hadron Collider

The Large Hadron Collider (LHC) is the largest currently operating particle accelerator and offers arguably the best chance at finding new physics at high energies. Built in the tunnel of the former Large Electron Positron Collider (LEP), it is capable of achieving over 60 times the center of mass energy that LEP reached at its peak. Largely thanks to this increase in energy that allowed for the discovery of the Higgs boson in 2012 [27, 28]. What made this possible was the switch to accelerating protons instead of the electrons and positrons. Since protons are more massive, one is able to impart more kinetic energy onto them before they so fast that losses synchrotron radiation prevent further acceleration. However, this also comes with a trade-off. Unlike electrons, protons are not elementary particles, but rather consist of three valence-quarks alongside a varying amount sea-quarks and gluons. Each of these particles carries a fraction of the total proton energy, the distribution of which is described by the parton density function or pdf. This means the actual center of mass energy in a collision of two proton constituents can be anywhere between zero and the full 13 TeV. So, unlike with a electron positron collider where one could precisely adjust the collision energy and scan along it in oder to accurately measure resonances, the LHC produces collision with a wide range of energies. Additionally, electrons only interact via QED and weak processes, which allowed LEP to perform high precision electro-weak measurement. Protons, or more accurately quarks and gluons, on the other hand, can also interact through QCD. Since QCD is inherently more strongly coupled than both QED and the weak interaction at LHC energies, these QCD processes will be dominant. Fig. 1 shows the calculated cross sections for different final states at proton/proton colliders. As one can see from this, the total cross section σ_{tot}, which is mostly made up out of soft QCD processes, is many times greater than that of any electro-weak process such as W or Z production. Even if we introduce a $p_{Tjet} > 100$ GeV cut on the jets, the jet cross section is still 10 times greater the the electroweak one. This results in a large background of QCD events, making exact identification of particles and processes even more vital for LHC precision measurement.

2.2 From Detector to Calorimeter Image

The particles produced at a collider cannot be directly observed. They can, however, be indirectly observed based on the their interactions with the detectors surrounding the collision point. The four main experiments at the LHC, ATLAS, CMS, ALICE and LHC-b all have different setups and, in the case of ALICE and LHC-b, also vastly different goals. However, they all share the same underlying principles, which we will now discuss, using the CMS-experiment as a basis.

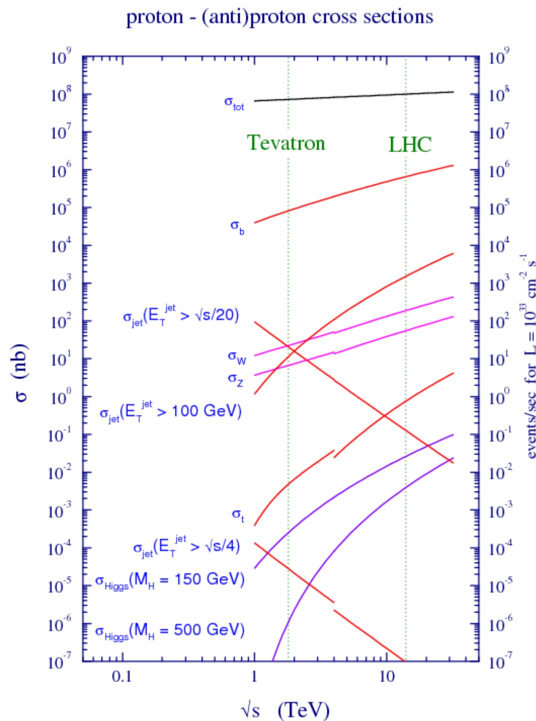

Figure 1: Cross sections for proton-(anti)proton colliders, taken from [29]. The cross section for jets is significantly larger than for any electro-weak process, even after a $p_{Tjet} > 100$ GeV cut.

Fig.2 shows a slice of the CMS detector. Particles originate form the interaction point, where the two proton beams cross, in the very left of the figure. The first part of the detector the particles encounter is the tracking system, indicated by the concentric rings on the left of Fig.2. Designs of the tracking system vary from experiment to experiment, but their function generally consist of recording the path of particles without absorbing a large portion of the their energy in the process. Since the detector is permeated by a strong magnetic field, these tracks will be bent for charged particles. Based on the radius of these bends one can reconstruct the transverse momentum component of a particle. After the tracking system, the particles reach the calorimeter, usually split into an electromagnetic and hadronic section, represented by the green and yellow areas in Fig. 2. The electromagnetic calorimeter (E-cal) is designed so particles like electrons and photons deposit their energies in the calorimeter in the form of an EM-shower. Hadrons, on the other hand, will pass trough the E-cal, losing only a small fraction of their energy, until they reach the hadronic calorimeter (H-cal), where they, in turn, cause a hadronic-shower. The amount of energy a particle deposits in both E- and H-cal can be used to determine the particle type as well as its original energy. Most standard model particles will not reach beyond the calorimeters. The exception to this are muons, which are

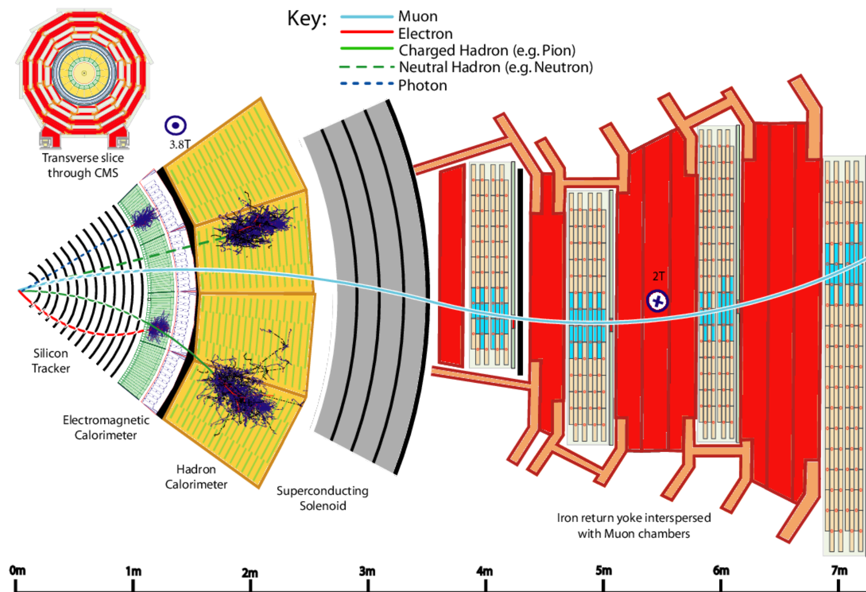

Figure 2: Schematic view of a transverse slice of the CMS detector, with interactions of particles with specific parts of the detector, taken from Ref. [30].

massive enough to pass trough the calorimeters with a significant part of their energy intact. This is when they reach the muon system, indicated by the thin orange stripes on the right in Fig. 2. The muon system is essentially a larger, less granular tracker, recording the path the muons take. Since muons do not deposit their energy in the calorimeters, the only way of reconstruction their energy is using the bend of their tracks. However, most muons are highly energetic leading to only slightly bent tracks. For this reason the large coverage of the muon system is needed to accurately reconstruct the bend radius.

With this we now have a set of tracking, calorimeter and muon information about each event. This means the most vital information for our machine learning analysis comes from the calorimeter. Since we want to use the power of image-classification networks, we now need to translate this calorimeter information into an, ideally 2 dimensional, image. For this purpose we first need to introduce two variables. The first is the azimuthal angle around the beam-pipe, ϕ. The second is the pseudo-rapidity η, defined as

$$\eta = -\ln(\tan(\frac{\Theta}{2}));, \tag{1}$$

where Θ is the angle between the beam-pipe and the position in the detector. The left side of Fig. 3 illustrates the positions of ϕ and Θ in the detector cylinder. We can now calculate η and ϕ for each calorimeter segment and map the energy deposited in them onto into a 2 dimensional heat-map in the η-ϕ plane, as demonstrated in on the right half of Fig. 3. Using this, we can translate LHC events into a form that an image based neural network can use.

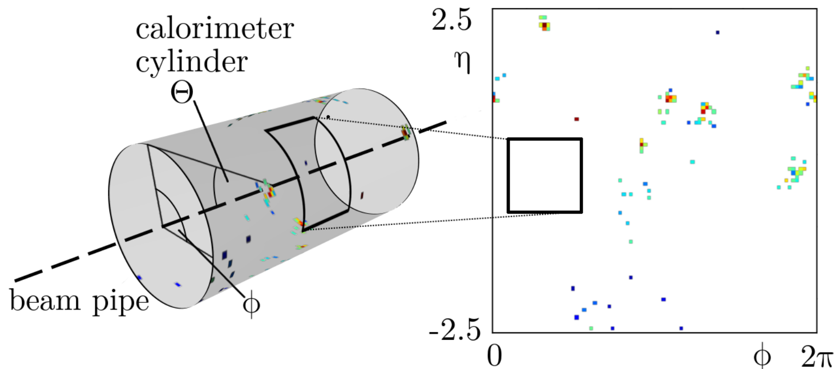

Figure 3: Example of mapping the calorimeter cylinder onto a η-ϕ heat-map.

2.3 Jet Definitions

Quarks and gluons produced in particle collisions are not directly observed. Due to the collinear divergence of QCD [31] these particles will emit quarks and gluons under all angles, with most of the particles being emitted in the soft collinear limit. During this splitting the partons will lose energy, until they reach an energy scale where, because of quark confinement, the partons will form hadrons, usually pions or other mesons. The resulting collimated stream of stable particles is called a jet. These jets can be seen as a stand-in for the original partons produces in the collision.

However, the actual definition of whether a specific particle belongs to a jet or not is not clear cut. Especially if there are several jets in close proximity, deciding to which jet a particle should be attributed to becomes highly difficult. For this reason several algorithms have been developed over the years that take a set of particles and cluster them into jets. The most commonly used of these are the generalized k_T algorithms, most importantly the anti-k_T algorithm [32], that we will now look at in detail.

Generalized k_T algorithm: Let us assume we have a set of particles n with transversal momenta k_{Ti}, pseudo rapidity η_i and azimuthal angle ϕ_i, $i \in n$. Then we define two distances,

$$d_{ij} = \min(k_{Ti}^{2p}, k_{Tj}^{2p}) \frac{\Delta R_{ij}^2}{R^2} \ , \tag{2}$$

where R is the freely chosen jet radius and ΔR_{ij} is the distance in the η-ϕ-plane,

$$\Delta R_{ij} = \sqrt{\Delta\eta^2 + \Delta\phi^2} \ , \tag{3}$$

and the beam distance

$$d_{iB} = k_{Ti}^{2p} \, . \tag{4}$$

Then one looks trough both lists for the smallest distance. If it is a distance between two objects d_{ij}, one removes i and j from the set of particles, recombines them into a new object and then adds this object to the particle set. However if the smallest distance is found in the d_{iB} list, one can assume that there is no more particle close enough to i to be recombined. In this case the object with the smallest distance is removed and classified as a jet. After each search for the smallest distance, the distance lists are recalculated with the remaining objects in the particle set, after which another search is performed. This process is repeated until the set is empty.

Depending on which value one chooses for the exponent p, this actually represents different algorithms. If $p = 1$ it is the k_T-algorithm, for $p = 0$ the Cambridge/Aachen algorithm and finally one has the anti-k_T algorithm if $p = -1$.

2.4 Top and QCD Jets

One major difference between the jets produced by top quarks and those originating from lighter quarks stems from the fact that up, down, charm, strange and bottom quarks all hadronize and are therefore able to reach the detector, while tops decay before that. This means that the particles in light jets all originate from QCD mediated soft collinear splitting, however in a top jet the electroweak decay of the top quark

$$t \to W^+ b \tag{5}$$

$$W^+ \to u\bar{d}/c\bar{s}/\bar{l}\nu_l \tag{6}$$

plays an important role. For this reason we will refer to non-top jets as QCD jets, despite the fact that QCD interactions are still relevant in the formation of a top jet.

Hadronic top decays in particular have a significant effect on the substructure of the top jets. Since one effectively ends up with three independent quarks, each forming their own jet. This results in a distinct three-prong structure, that can be used to differentiate them from QCD jets. Whether or not these three sub-jets are viewed as a single fat-jet or as three individual jets depends on two things. For one, the momentum of the decaying top quark, because a high momentum will cause the three sub-jets to be very collimated. Further, it also depends on the jet radius used in the jet clustering algorithm.

3 Neural Networks

The underlying idea of machine learning is to generate a model, in our case a neural network, to describe a set of data points. This model is made up out of several **neurons**, grouped together into layers, and can in principle be imagined as a complex function with a large number of parameters. These parameters are then modified in a gradient descent process in order to fit the model to the data points.

3.1 Neurons and Layers

A neuron is the basic component of a neural network. A nice way of imagining a neuron is as a node in a computational graph. By itself it simply represents a scalar value on which computational operations are performed. How these nodes are interconnected is in principle a free choice. In practice we define layers of neurons to perform commonly useful operations.

3.1.1 Dense Layers

Dense layers or fully-connected layers are a basic building block of neural networks. Several more complex layer types, among them also the capsule layer we will use later on, are essentially evolved out of dense layers.

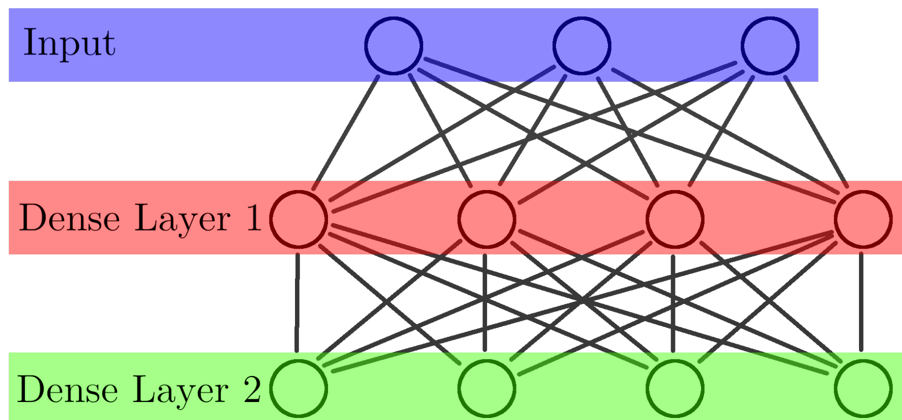

Figure 4: Schematic view of two connected dense neural network layers with four neurons and a three neuron input layer.

Each dense layer consist of a variable number of neurons, however, in order to be well defined, a dense layer needs to be connected to either some form of input layer or to another dense layer. Fig. 4 shows one such connected setup. Let us initially look at dense layer 1 (red).

Each neuron in this layer receives a value from each input neuron (blue), and in turn passes its output value to each neuron in the following dense layer (green). However these contributions, indicated by the gray lines, are weighted, with each line having a different, trainable scalar weight. In order to determine the output of a neuron, one needs to add up all weighted contributions as well as the so called bias term. The bias in this case refers to an additional number of trainable parameters equal to the number of neurons in the layer. Mathematically the operation of a dense layer can be expressed as a vector multiplied with a matrix. Let us assume we have an input layer with N neurons and values $i_{1..N}$ and want to connect it to a dense layer with N' neurons and values $j_{1..N'}$. In this case we define a $N \times N'$ weight matrix W with entries $w_{1..N,1..N'}$ and the bias vector $\vec{b}$ with entries $b_{1..N'}$. Now we can calculate the output of the dense layer using:

$$j_{n'} = b_{n'} + \sum_{n=1,..,N} i_n w_{n,n'} \ . \tag{7}$$

3.1.2 Pooling Layers

Unlike the other layers discussed in this section pooling layers do not have any trainable parameters and always perform the identical operation on their given input. The main purpose of pooling operations in machine learning is to reduce number of parameters one has to deal with, ideally by removing irrelevant information. Generally this is done by reducing the resolution of either the input image or an intermediate network result that has an image-like structure, usually the output of a convolution. What the pooling specifically does is split the image into sections with a $n \times m$ pixel size, $n, m \in \mathbb{N}$. This $n \times m$ size is also know as the size of the pooling kernel. Then all pixel values in a section are reduced down to one number, by some mathematical operation, most commonly either by taking the maximum of all pixels in a section (max-pooling) or by taking the average (average-pooling). An example of both these types of pooling with a 2×2 kernel is shown in Fig. 5. Here the large 6×6 matrix is the input, the small 3×3 matrix on the bottom left is the output of a max-pooling operation and the matrix on the right is the output of a average-pooling operation. The blue and red squares indicate the pooling-kernels. Note how using a 2×2 kernel reduces the total amount of pixels in the output to a quarter of the number in the input.

One common criticism of pooling is that by removing information one loses potential classification power, however many times pooling is necessary, because a network designed to handle all the information would be too big to be trained within a reasonable time.

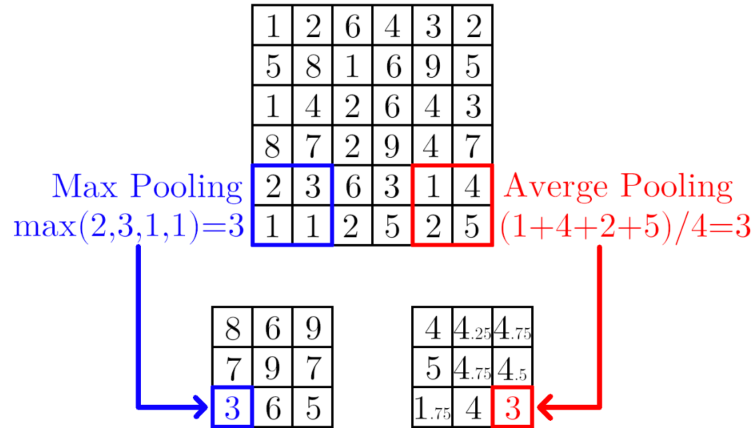

Figure 5: Demonstration of the effects on pooling on the input image (top), split into max-pooling (left) and average-pooling (right).

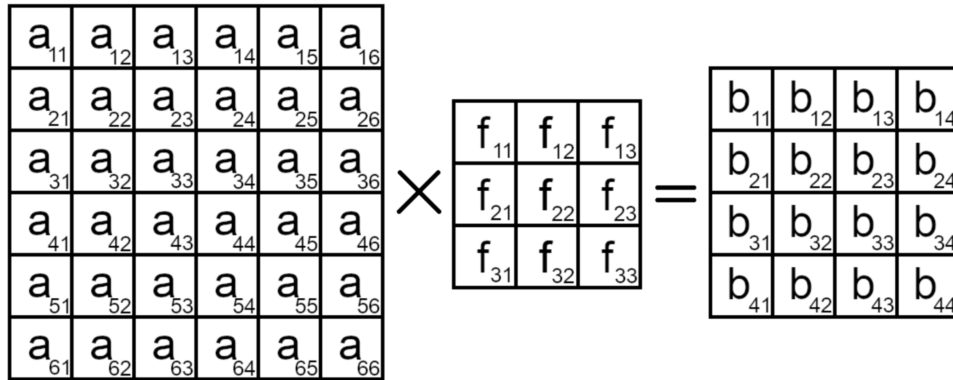

Figure 6: Schematic of a 3×3 convolution kernel applied to a matrix.

3.1.3 Convolutional Layers

Convolutional layers are based on convolution filter operations, which have been employed in image processing for a long time. The underlying idea is to define a filter matrix (or kernel) with size $N \times M$ and then perform an element wise multiplication of the filter and an $N \times M$ section of the image one wants to process. This can be seen in Fig. 6. Here we have a 3×3 kernel, an input matrix A with elements a_{ij} and an output matrix B with elements b_{ij}. In order to get the first output entry b_{11} we take the element wise product between the kernel and the sub matrix $A[1, 2, 3; 1, 2, 3]$ and sum over the individual results. So:

$$
\begin{aligned}
b_{11} = {} & a_{11}f_{11} + a_{12}f_{12} + a_{13}f_{13} \\
& + a_{21}f_{21} + a_{22}f_{22} + a_{23}f_{23} \\
& + a_{31}f_{31} + a_{32}f_{32} + a_{33}f_{33} \; .
\end{aligned} \tag{8}
$$

This kind of operation allows us to not only look at individual pixels in an image, but at groups of close by pixels, thereby enabling us to search for patterns. A classic example of this would be a so called edge detection filter as show in Fig. 7. As one can see, if all values in a 2×2 input section are equal, the output is just 0, however if there is a vertical gradient the output will be different from zero, with the sign indication the gradients direction and the absolute value describing its steepness. The end result is a image where sharp edges are highlighted. Commonly such an image is referred to as a feature map, since it is a map describing certain features of the images, the feature in this case specifically being edges.

More advanced kernel structures can be used to extract more complicated features. However, figuring out which exact kernel is optimal for a certain set of images is a difficult problem. The machine learing approach to convolutions therefore consists of only choosing the size of the filters and having the actual entries be trainable weights. This effectively allows the network to build its own filters in order to search for the specific features the network deems to be most important for the task at hand.

The simplest convolutional layer therefore takes an input of images, performs a convolution operation using the trained kernel and then output one feature map. It is, however, common practice to use more than one filter, resulting in an output consisting of multiple feature maps. This is done to allow the network to extract multiple features simultaneously.

Furthermore, one usually employs several consecutive convolutional layers. The idea behind this is that the first convolutions will extract basic features, like edges, and subsequent layers will then use these to build more complex features.

One rather recent development is to employ so called stride convolutions. Unlike in a standard convolution layer where the kernel is moved one pixel position at a time, the stride variant moves the kernel are by a customizable stride number. This will drastically reduce the resolution of the convolution output, thereby acting similarly to a pooling layer. Fig. 8 shows an schematic representation of a strided convolution, with a horizontal stride of 2 and a vertical stride of 3. The blue frame indicates the starting point of the convolutional kernel, the green and orange ones show its position after one step in the horizontal or vertical direction, respectively. The colored pixels in the 3×2 output belong to the convolutional operations preformed at the position of the respectively colored frames. It is once again of note how the size of the input is drastically reduced. Compared to a rigid pooling operation, stride

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

Figure 7: Example of a specialized convolutional kernel designed for vertical edge detection.

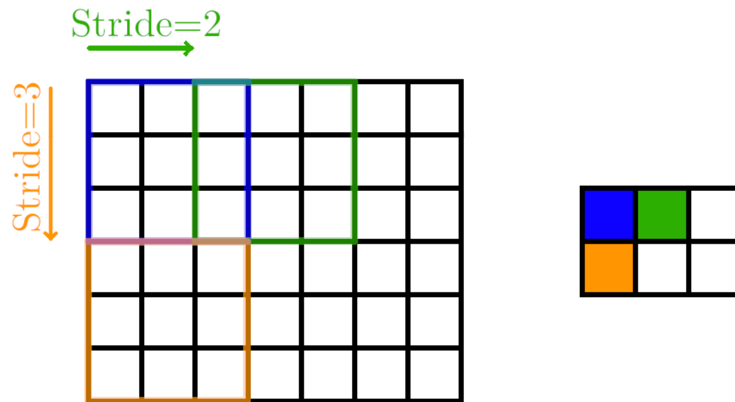

Figure 8: Illustration of different stride lengths in a convolutional operation, in this case vertical stride = 2, horizontal stride = 3.

convolution layers should, in principle, offer the advantage of having trainable parameters and therefore being able to adjust based on the networks need. Throughout our work in capsule networks we employ both pooling layers as well as stride convolutions, however we found little performance difference between the two, which is in agreement with other investigations into pooling vs strides [33].

3.2 Activation Functions

A network made up entirely out of layers described above is equivalent to a series of linear matrix multiplications, this means that by itself it will have trouble describing non linear problems. That presents a problem since most physical problem one would want to use a neural network on are not linear. Therefore we need to introduce some source of nonlinearity

to our system. In practice this is achieved by so called activation functions. These non linear functions are applied to each layer output.

The two most commonly used activation functions are the Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \max(x, 0) \ , \tag{9}$$

and sigmoid:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \ . \tag{10}$$

The shape of both of these functions in shown in Fig. 9. Sigmoid is generally used for the final layer of a network, where it ensures that the outputs are constrained between 0 and 1, a behavior especially desired in background vs. signal classification tasks, since this allows the network output to be interpreted as a probability. Using sigmoid in intermediate layers, however, comes with problems, since, as can be seen in Fig. 9, the function quickly converges to 1 or 0 for large or small inputs, respectively. This can lead to vanishingly small gradients during network training, either greatly increasing the time needed for the network to converge or even causing the training to fail completely. For this reason ReLU is usually the preferred activation function of the two for intermediate layers. By having a linear component it does not suffer from the vanishing gradient problem for positive values, while still providing some form of non-linearity through the cut-off at $x = 0$.

3.3 Loss Functions

Now that we have the basic building blocks of our network, we want to actually start training it. To this end we need a metric to judge how well the network is learning. Loss functions fulfill this purpose, generally speaking they indicate how well the current network is able to describe the date. There is a large number of possible loss functions, one commonly used

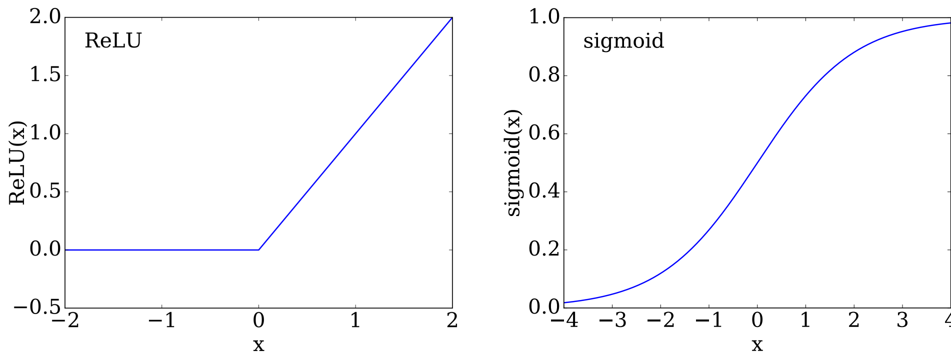

Figure 9: Left: Plot of ReLU activation function. Right: Plot of sigmoid activation function.

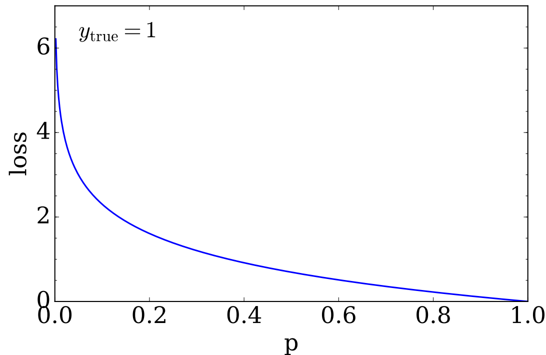

Figure 10: Shape of the cross entropy output for different prediction values p and a truth label value of $y_{true} = 1$.

in classification tasks is cross entropy. To examine this more closely let us assume we have a network that is supposed to classify images into signal (label $y_{\mathrm{true}} = 1$) and background ($y_{\mathrm{true}} = 0$). The simplest way of achieving this is to build a network with a single output neuron with a sigmoid activation function. The network output p is then constrained between 1 and 0, and similar to the definitions of y_{true} we can call predictions closer to 1 signal and ones closer to 0 background. We can then define the cross entropy as

$$L_{CE} = -(y_{\mathrm{true}} \log(p) + (1 - y_{\mathrm{true}}) \log(1 - p)) \ . \tag{11}$$

Since the label y_{true} is always either 1 or 0 the first or second part of eq. 11 is zero. If one for example takes a signal event, the cross entropy becomes

$$L_{CE} = -1 \log(p) \ . \tag{12}$$

This is shown in Fig. 10, where we plot the predicted value p against the loss value from eq.12. We can see that the loss gradually decreases, the closer the prediction gets to the correct value of 1.0. The behavior for a background images is identical, save for the fact that Fig. 10 would be mirrored along the y-axis. In practice one groups several input images into batches, which are processed simultaneously by the network. This allows us to calculate the loss for each image in the batch and, from this, get an average loss. This combined loss gives us a good estimation of how well the network performs; if most predictions are correct the loss will be small, however if the network decisions are false we will see a large loss. Therefore we can optimize our network by minimizing the loss.

3.4 Gradient Descent

Gradient descent is the method used minimize the loss and thereby train the network [34]. We assume we have a dataset X and a network-model $m(\Theta)$, which is dependent on a set of parameters Θ, and finally we have the loss-function $L(X, m(\Theta))$. Initially Θ will be randomly initialized, now we want iteratively adjust it until $m(\Theta)$ describes X sufficiently well. To this end we calculate the gradient of $L(X, m(\Theta))$ with respect to each parameter in Θ and then modify each of these parameters accordingly. One way of doing this is a first-order gradient descent. Here we first calculate said gradients:

$$v_t = \eta_t \nabla_\Theta (L(\Theta_t)) \; . \tag{13}$$

Where η_t is the learning rate (LR) and t indicates the current iteration step. Following this we update Θ:

$$\Theta_{t+1} = \Theta_t - v_t \; . \tag{14}$$

The choice of the learning rate has a large influence on how well this method works. If η_t is to small, the network will either not converge at all or take an unreasonable amount of time to do so, while too large a learning rate may cause the training to overshoot the loss minimum we want to reach. Another potential downfall of gradient descent is saddle points, where the gradient will be nearly zero, causing the training process to slow down to a crawl.

One way to improve the gradient descent method has previously been hinted at, while we discussed batch-averaged loss functions. The main idea is to split the data set X into small batches B_k containing M data points. If the full data set has $\#X = n$ we will end up with $k = 1, .., \frac{n}{M}$ such batches. Now, instead of using all of X to calculate the gradient, we use one of the batches instead. Since the batches are effectively a random selection of data points, the gradients will differ slightly between each batch. This method, also known as stochastic gradient descent (SGD) [35], introduces some stochastic variance and thereby makes the algorithm more stable. Further, calculating the gradients of $\frac{n}{M}$ M-sized batches requires less computational resources than calculating them for all n data points at once, meaning that the batch structure both speeds up the training process and allows us to train large networks with big datasets without running into memory constraints.

Another way to improve the gradient descent method is the addition of a momentum term [36, 37],

$$v_t = \gamma v_{t-1} + \eta_t \nabla_\Theta (L(\Theta_t)) \tag{15}$$

$$\Theta_{t+1} = \Theta_t - v_t \; . \tag{16}$$

The term $v_t = \gamma v_{t-1}$ with $\gamma < 1.0$ can be seen as a recursive expression of a running average over all previous gradients, with the most recent gradient being weighted by γ, the one before that by γ^2 and so forth. A momentum term like this has several advantages, for example it allows the training to gradually pick up speed even in an area with a comparatively flat gradient. Further, it also make the algorithm less vulnerable to small oscillations in the gradient.

However, even with both stochastic gradient descent and momentum one problem remains. Namely that, in an ideal case, one would not have a static learning rate, but an adaptive one. One way to approach this is to design a learning rate schedule, which starts out with a large learning rate, allowing the network to quickly find a rough minimum, and then gradually decreases that rate so the network can slowly approach the exact minimum without overshooting it. Methods like this are often employed, usually implemented as a exponential decay of the learning rate. However, even that is still a rather static way of modifying the learning rate, since it is identical for every training run and also affects all directions equally. A better method would be to get information about the current curvature in all directions and adapt the learning rate in a way that gives a high rate for flat directions, and a low rate for steep ones. This would require us to calculate the second derivative of the loss function, and scale the learning rate by its inverses, so $\eta = \eta_0 \left(\frac{\partial^2 L}{\partial \Theta^2}\right)^{-1}$. Since we have significantly more than one parameter in our model we would have to calculate the Hessian matrix instead of the simple derivative. This simply is not computationally feasible, even using batches. One way, however, to approach the behavior we could get by using the Hessian, is to instead scale the learning rate by the second moment (or un-centered variance). This is done in the Adam optimizer [38] which is also the optimization algorithm used in the remainder of this thesis.

The main idea of Adam is to keep a running average over both the gradient as the momentum term (m_t) and the gradients square (s_t) as an estimate of the second moment, defined by

$$g_t := \nabla_\Theta(L(\Theta_t)) \tag{17}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{18}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) g_t^2 \ . \tag{19}$$

Here β_1 and β_2 are separate decay constants, defining how long previous results are remembered. Default values for these constants are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. A detail of Adam is that m_t and s_t are bias corrected before updating the model parameters, by calculating

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{20}$$

$$\hat{s_t} = \frac{s_t}{1 - \beta_2^t} \ , \tag{21}$$

and then used to update the parameters according to

$$\Theta_{t+1} = \Theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon} \,, \tag{22}$$

where $\epsilon \approx 10^{-8}$ is a constant introduced to stop the expression from diverging, should the remainder of the denominator get to small.

3.5 ROC Curves and Classifiers

When trying to quantify and compare the performance of neural networks one of the best tools available is the so called receiver operating characteristic or ROC curve and its area under curve or AUC. The underlying idea of the ROC curve is to plot the false positive rate FPR against the true positive rate TPR. In physics terms they are therefore equivalent to the miss-tagging rate $1 - \epsilon_b$ and the signal efficiency ϵ_s respectively. The TPR describes how much many of the signal events passed to the network are correctly identified as such, while the FPR indicates the fraction of background events mistakenly called signal. Ideally one wants a large TPR while keeping the FPR small.

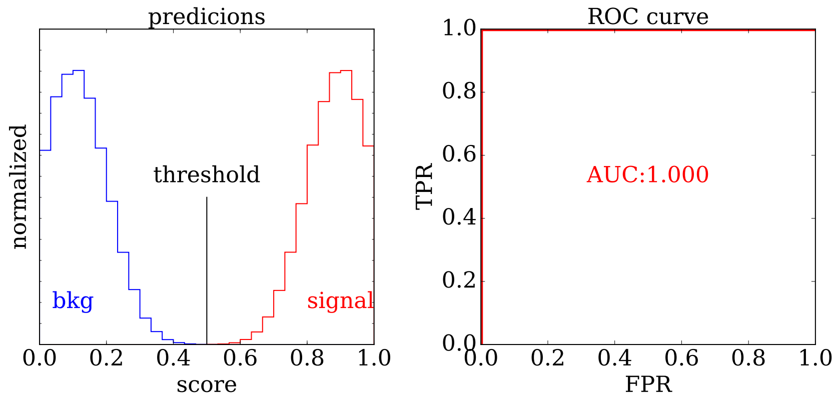

Figure 11: Left: Distribution of prediction scores for signal (red) and background (blue) for perfect classifier, threshold is moved along the x-axis to scan for TPR and FPR. Right: ROC curve shape for perfect classifier

We will now examine how to produce these rates from a neural network output and how to interpret the resulting curve using a simple example of signal versus background classification. Let us assume our evaluation sample contains an equal amount of signal and background events. When passed to the network it will generate a prediction score, usually constrained within $[0, 1]$. An ideal network will be able to split signal and background into two distinct distributions with no overlap. This is shown in the left-hand side of Fig. 11. Now we can place a cut-off point or threshold on our x-axis and declare every event with a prediction score above

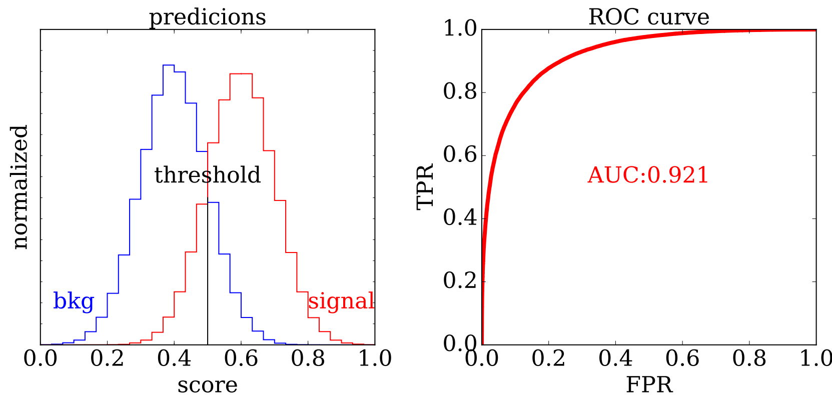

Figure 12: Left: Distribution of prediction scores for signal (red) and background (blue) for imperfect classifier, threshold is moved along the x-axis to scan for TPR and FPR. Right: ROC curve shape for imperfect classifier.

this threshold as signal and everything below as background. In order to produce the TPR and FPR we initially place the threshold at 0 and gradually increase it. With the threshold at 0 everything is classified as signal. This results in both TPR = 1.0 FPR = 1.0, which is also our first point in the ROC curve, shown on the right side of Fig. 11. As we now increase the threshold less and less background will be labeled as signal, causing the FPR to decease, while the TPR stays 1.0. In the ROC curve this is represented by the line at the top with a constant TPR = 1.0. This continues until the threshold hits 0.5, at which point we get TPR = 1.0 and FPR = 0.0, indicated by the top-left corner of the ROC curve. When we now further increase the threshold the FPR will remain 0.0, however now the TPR will begin to fall and when the threshold finally reaches 1.0 both the TPR and FPR will be equal to 0.0, since no events are classified as signal anymore.

Now that we have our ROC curve we can interpret it. Generally speaking the area of most interests is the top-left corner. If the curve fills this corner perfectly it means there is a threshold value for which the FPR is 0.0 and the TPR is 1.0, indicating a perfect classifier. In order to quantify how close the curve gets to this point we can calculate the area under the ROC curve, the previously mentioned AUC. If the AUC is 1.0 the top-left corner is completely filled, if its smaller than 1.0 we know that our classifier is not perfect. One such example is shown in Fig. 12, here the prediction score distributions of signal and background overlap, meaning we have no threshold position where signal and background are perfectly separated, resulting in a curve with a different shape and an AUC < 1.0. This is the usually case in practice, as an AUC of 1.0 is almost never achieved unless the problem was already rather simple.

Finally we can examine the case where the classifier cannot distinguish between signal and background and their prediction score distributions are virtually identical. This is shown

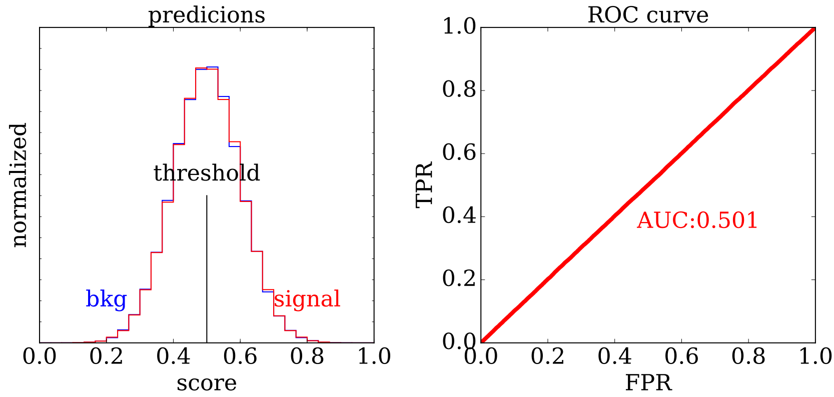

Figure 13: Left: Distribution of prediction scores for signal (red) and background (blue) for non-functioning classifier, threshold is moved along the x-axis to scan for TPR and FPR. Right: ROC curve shape for non-functioning classifier.

in Fig. 13. Here the TPR and FPR are the same for each threshold position, causing the ROC curve to be a straight line staring at $(0,0)$ and ending at $(1.0,1.0)$ and resulting in an AUC of 0.5. This is the lowest AUC value one runs into in practice, and it usually indicates that the network did not learn anything and is just making random guesses. Interestingly a AUC of less than 0.5 would mean that the network consistently identifies signal as background and the other way around, possibly indicating an error in the labeling of the sample.

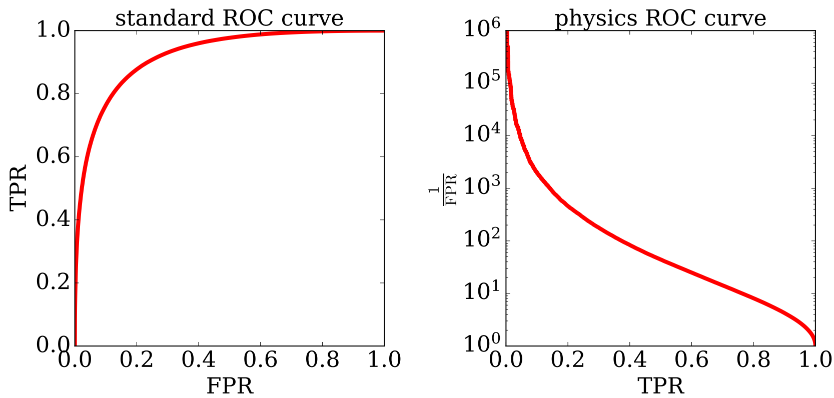

Figure 14: Comparison between standard ROC curves (left) and the physics ROC curve used in this thesis (right).

For our physics use-case we have however found out that not only the AUC is important when evaluating a networks performance, but also the shape of the ROC curve. It turns out, for example, that the working point at a TPR of 0.3 is often interesting for our signal and background rates. However if we look at the right side of Fig. 13 we see that reading of this working point proves to be rather difficult. In order to remedy this we will be using a different

style of ROC curve through the rest of this thesis. The first thing we change is switching the two axis, so we now have the TPR as our x-axis. Further we no longer simply plot the FPR, but instead $\frac{1}{\text{FPR}}$, using a log scale. A comparison between the two is shown in Fig. 14, with the standard ROC curve on the left, and our modified curve on the right. Now reading of and comparing the FPR at the TPR $= 0.3$ working point is a lot clearer.

4 Capsule Network

4.1 Motivation

In order to understand the advantages of the capsule network setup it is instructive to first look at how standard convolutional neural networks (sCNN) operate, and more importantly why they have trouble classifying full events. Fig. 15 shows the architecture of one such sCNN. In essence they can be split into a convolutional and a dense part. The first is a series of convolutional layers and pooling layers, which extract progressively complex features from the input image. Their output consists of a series of prediction scores, each associated with different features and sections of the input image. These prediction scores then represent a probability of the respective feature being present in the certain image section. The second part of network consists of several dense layers, shown on the very right of Fig. 15, these take the prediction scores from the convolutional part and use the presence or lack of specific features to make a final classification prediction.

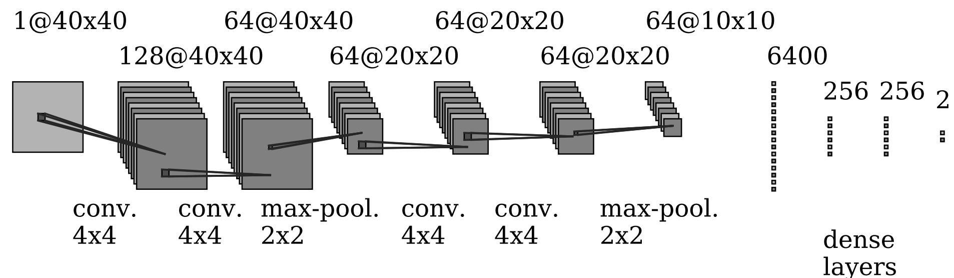

Figure 15: Architecture of a standard convolutional neural network, in this case based on an architecture form Ref. [22].

The way an sCNN operates is best illustrated using an example. Let us assume we want to use a network to decide whether an image contains a face. Since this is an image classification task we know from previous experience [20] that using convolutions is promising approach.

Fig. 16 shows a potential input image for such a network (left) and an idealized example of how the convolutional part could process said input (right). In this case, important features that should be present in the input in order for it to be identified as a face could be eyes, mouth, nose as well as the generally round shape of the face itself. The percentages next to the marked features represent the prediction scores, indicating how confident the network is that the object in the marked location is actually a eye, mouth or nose.

However, as excerpts from a potential training sample in Fig. 17 show, such a sample will likely include a lot of faces with different shapes and topologies, causing the network to learn that the absolute position of the individual features may vary, meaning it does not matter

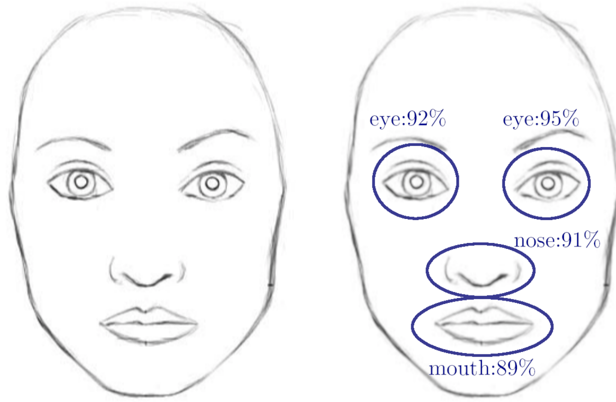

Figure 16: Facial features as extracted by a convolutional network, percentages indicate prediction confidence whether image contains specific features at this position. Original taken form Ref. [39]

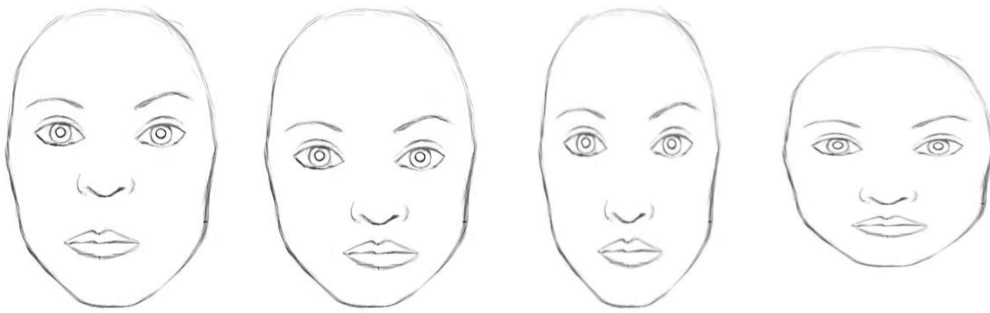

Figure 17: Examples of faces with different absolute and relative feature positions. Original taken form Ref. [39]

whether the eyes are at a specific position in the image of a few pixels further to the left. In addition to this, even if the potions of features turned out to be important, the dense part of the network would unable to comprehend positional relations between the individual features, since the only information about the features it receives is how likely they are to be present, and no descriptor of their position. This initially seems counterintuitive, since it should be able to infer the position of features based on the convolutional output, after all different convolution outputs are associated with different input image section. To understand why this does not work we need to take a look at the transition from the convolutional to dense part. As can be seen in Fig 15 the final convolution output consist of 64 feature maps with a 10×10 pixel resolution. This then gets reshaped into an one dimensional array with 6400 entries, better imagined as 64 sets of 100. Now each of those 64 sets corresponds to one feature, and each of the 100 entries within those sets represents approximately a 4×4 section of the original image, since we perform two pooling operations with a 2×2-kernel. Of course the structure of the

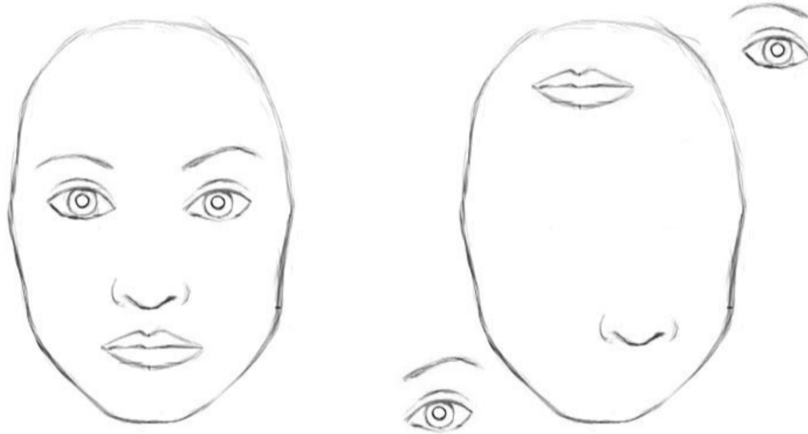

Figure 18: Demonstration of the problem with standard convolutional neural networks, because both images have the same features present, both are classified as faces. Taken form Ref. [39]

convolutions causes each entry to be associated with more than just the 4×4 section, however for our example this is a close enough approximation. The value of each entry describes the confidence that the relevant feature is present in that part of the image. Therefore the dense layer is capable of determining where features are located within the image based on which of the 100 input entries has a high value. However for the dense layer the order of the inputs is irrelevant, since it treats each input independently. This means that the standard convolutional setup cannot encode relative distances between features, a property which becomes relevant in a case like Fig. 18. Here we have two images containing the exact same features, but in one case they are shuffled around, resulting in an image quite different from a normal face. However, since they do have the same features, a standard CNN will classify them both as a face.

In a normal use case this is not a huge problem, as one's facial recognition software will most likely never run into an image as in Fig. 18, but for full event tagging this becomes a lot more relevant. One of these cases is show in Fig. 19. Here we have two images, each containing the same two top jets, which are rated with the same prediction score. However, in one of the images the jets are back to back, indicating a simple production of a top-pair via a QCD process, while the other image has two highly boosted jets with an asymmetry in ϕ, pointing towards additional invisible particles being involved in the process, resulting in missing transverse energy. In other words the two images belong to vastly different processes, yet for a standard convolutional network the images once again look nearly identical, since they contain the same features. This is one of the biggest hurdles we have to overcome before we can do image based event classification and one way of doing this is by using capsule networks.

The main idea of capsule networks is to no longer only use scalar values in your network, but to use vectors. So instead of describing your features using only single prediction

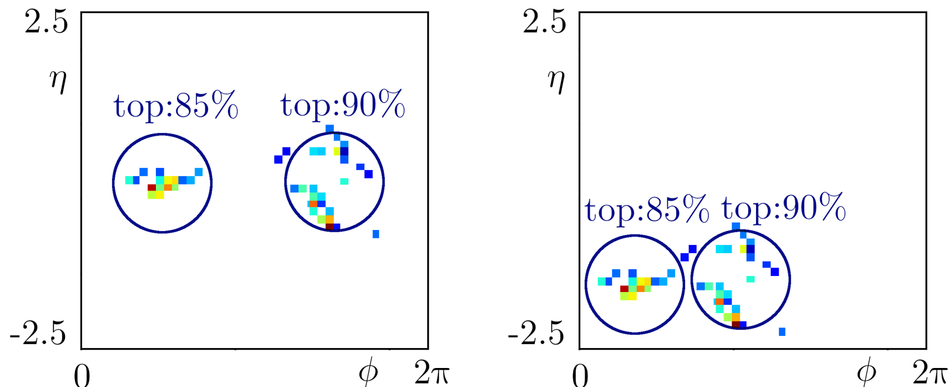

Figure 19: Further example of the sCNN problem, once again features are identical, but underlying events are different.

scores, one can now use n instantiation vector, the entries of which represent properties of the feature. Fig. 20 illustrates an example of this using the previous two top jet images as a base. Now instead of having one single prediction score for the two jets we have a vector encoding properties like the p_T, m, its η and ϕ position. This will ideally allow the network to learn the spacial relations between features that are necessary to tag whole events. The specific features mentioned here are just examples, chosen because they can easily be understood by physicists. When training an actual capsule network there is no reason why the features it teaches itself should correspond to our commonly used set of physical observables. They could easily be seemingly random linear combinations of observables that encode the same information using a different basis, or they could be features that were previously not considered, but turn out to be useful for the classification task.

We do, however, still need something to fill the prediction scores role. This is where another key aspect of capsule networks comes into play, namely that the Euclidean length of an instantiation vector corresponds to the probability that the object the vector is describing is indeed a member of the vectors class. So a longer vector corresponds to a confident prediction, while a short vector represents a less sure one.

4.2 Capsule Layers

Our setup is largely based on a setup [40] for the original capsule paper [41]. The first step is to create two new layer types the primary caps layer and the routing layer.

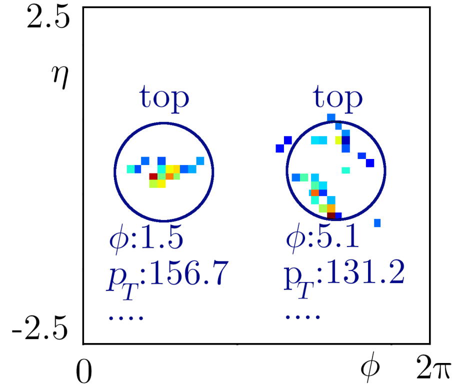

Figure 20: Example of how a capsule network could encode properties of top jets in a vector.

4.2.1 Primary Caps

The primary caps layer is not a classical neural network layer, in the sense that it has no trainable weights. The main point of primary caps is to take the output of the convolutional layers, consisting of a stack of feature maps, and to reshape it into something the actual capsule layers can use. This means the output has to be a series of vectors. When deciding which approach to take for this reshaping we need to remember an important principle of the capsule setup: Capsule entries describe properties of features. Ideally this is also true for the feature maps we get from the convolutional layers. Therefore, we take the first pixel of each map and define this stack of pixels as our first capsule, as illustrated in the top center of Fig. 21. This way, the entries of our new capsule relate to the features from the convolutional layers, with one capsule essentially being responsible for one section of the image. A slight problem arises, however, if we have more maps than dimensions in our first capsule layer. This can easily happen since one wants to have a large amount of feature maps in the convolutions to capture more information. However at the same time it is advisable to keep the initial capsule dimension small and then increase the dimensionality with each capsule layer, since this reflects the increase in complexity of the more elaborate features deeper layers are trying to describe. In order to fix this we simply need to ensure the amount of maps, n, is evenly divisible by the capsule dimension I, and simply split our stack of n identical-positioned pixels into $\frac{n}{i}$ individual capsules, as shown in the right of Fig. 21. Doing this for each pixel in our $X \times Y$-size feature-maps, we end up with $m \times X \times Y$ capsules. This is illustrated at the bottom of Fig. 21. For example if one has $n = 96$ feature maps with a 10×10 resolution and the desired capsule dimension is $i = 6$ one ends up with a total of $10 \times 10 \times \frac{96}{6} = 1600$ capsules.

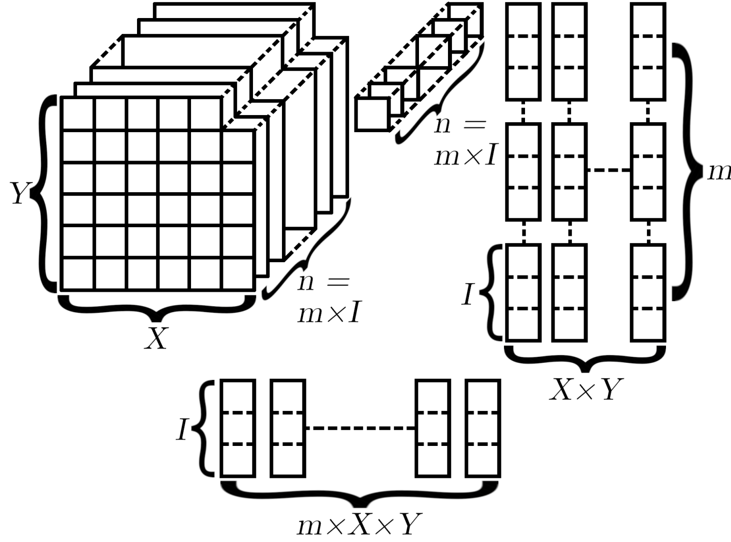

Figure 21: Illustration of how convolutional feature maps are reshaped into capsules. Top left: stack of n feature maps with resolution $X \times Y$. Top right: intermediate step, splitting feature maps into $X \times Y$ sets of m, I-dimensional vectors. Bottom: Capsule structure with $m \times X \times Y$ I-dimensional vectors.

4.2.2 Routing Layers

The routing layer is the heart of the capsule network and is where the classification decision making actually takes place. In a sense, it is to the capsule network what the dense layer is for a standard convolutional network. In order to understand how the layer functions in practice it is best to look at an example. Let us assume we want to have a routing layer which accepts J capsules with dimension I as its input and outputs J' I'-dimensional capsules. In the following we will call the input and output vectors $x_i^{(j)}$ and $v_{i'}^{(j')}$ respectively, with

$$i = 1, \;..\,, I \quad j = 1, \;..\,, J \quad i' = 1, \;..\,, I' \quad j' = 1, \;..\,, J' \;. \tag{23}$$

Fig. 22 shows this for the specific case of $J = 3$, $I = 2$, $J' = 4$ and $I' = 2$, however we do not lose any generality by using these numbers. Initially we calculate a set of intermediate vectors, indicated by the $J' = 4$ groups, each containing $J = 3$ vectors in the middle layer of Fig. 22. These vector are given by Eq. 24, where $W_{i'i}$ is trainable matrix of weights $w_{i'i}^{(j,j')}$. In general $I \neq I'$

$$u_{i'}^{(j,j')} = \sum_{i=1,..,I} w_{i'i}^{(j,j')} \, x_i^{(j)} \;. \tag{24}$$

At this point in a dense layer, one would now get the output by summing over all input contributions. If we did this in the capsule layer the end result would be equivalent to a dense

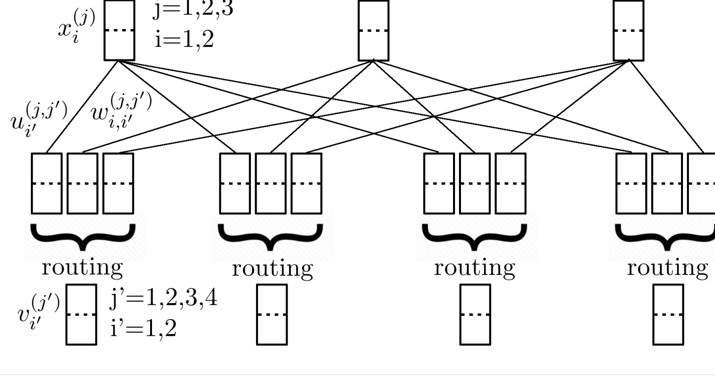

Figure 22: Capsule layer with three 2-dimensional input vectors and four 2-dimensional output vectors. Taken from Ref. [25].

layer with an unconventional input and output shape. However, we want to actually make use of the information encoded in both the capsule entries and their lengths. This is where dynamic routing is used.

4.3 Dynamic Routing

The routing algorithm's main purpose is to, as the name suggests, dynamically route the flow of information in the capsule network. We will first describe how exactly it operates, before explaining how and why it works. As the first step we define the $J \times J'$ matrix B, and set its components to $b_{j,j'} = 0$. Then we perform a softmax operation along j',

$$c^{(j,j')} = \text{SoftMax}_{j'}\ c'^{(j,j')} = \frac{\exp c'^{(j,j')}}{\sum_\ell \exp c'^{(j,\ell)}} \ , \tag{25}$$

giving us a set of weights $c^{(j,j')}$. Since the input to the softmax function were all zero, these weights are initially identical and equal to $\frac{1}{J'}$. Further they sum to one along the index used for softmaxing,

$$\sum_{j'} c^{(j,j')} = 1 \quad \forall j \ . \tag{26}$$

This normalizes the weights, by ensure they are constrained between 0 and 1. We now use these weights to combine the intermediate vectors $u_{i'}^{(j,j')}$ in a weighted sum:

$$v_{i'}^{(j')} = \sum_j c^{(j,j')} \ . \tag{27}$$

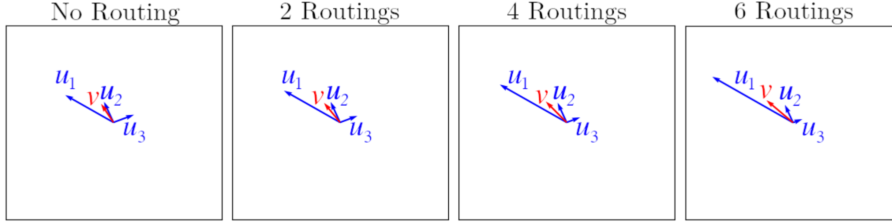

Figure 23: Example of the routing effects. Blue arrows are the input vectors, scaled by their respective weight, red arrows are the combined result. As seen in Ref. [25].

Further, we want to ensure that our output range is limited to the interval $[0, 1)$. To this end we apply the squashing function,

$$v_{i'}^{(j')} = \vec{v} \rightarrow \vec{v}' = \frac{\vec{v}^2}{1 + \vec{v}^2}\, \hat{v} \;, \tag{28}$$

to the result of the weighted sum. Here $\hat{v}$ is the unit vector in $\vec{v}$-direction. We will go into more details about the squashing in the following subsection. Since in the initial step all weights are equal, the first time we calculate this sum, it is effectively the average over all inputs. We now modify the weights based on the agreement between the individual $u_{i'}^{(j,j')}$ vectors and $v_{i'}^{\prime(j')}$.

$$b^{(j,j')} \longrightarrow b^{(j,j')} + \vec{u}^{(j,j')} \cdot \vec{v}'^{(j')} \;, \tag{29}$$

In oder to measure this agreement we use the scalar products between the vectors. If the vectors agree, they will be close to parallel, resulting in a large scalar product, but if they disagree the vectors will either be orthogonal or anti-parallel, causing a small or negative scalar product. After updating $b^{(j,j')}$ we once again perform the softmax operation on it. Using these weights we go back to Eq. 27 and recalculate $v_{i'}^{\prime(j')}$. This whole process is one routing pass, with $v_{i'}^{\prime(j')}$ being the corresponding output. We then repeat this process of updating the weights and recalculating $v_{i'}^{\prime(j')}$ for however many routings we want. In practice we typically need around 3 routing steps for good agreement.

The effects of the routing can be seen in Fig. 23, here the blue arrows symbolize the intermediate vectors $\vec{u}^{(j,j')}$ scaled with their respective weights $c^{(j,j')}$ and the red one represents the combined and squashed $\vec{v}'^{(j')}$, for $j = 1, 2, 3$ and any given j'. On the rightmost plot we can see the initial setup, with the other plots showing the effect of progressively adding more routing steps. The first thing we notice is that $\vec{u}_3$, which starts out orthogonal to $\vec{v}$ gradually has its length reduced, further the vector $\vec{u}_1$ is relativity parallel to $\vec{v}$, resulting in its weight getting enhanced. On closer inspection, however, one notices that the length $\vec{u}_2$, which is even more parallel to $\vec{v}$ than $\vec{u}_1$ barely changes, despite the fact that one would expect it to increase. This is because we use the scalar product for the reweighing, the result of which not only depends on the direction, but also the length of both vectors. This has two effects. For one it means long vectors will be weighted more, which is very much desired, since as previously

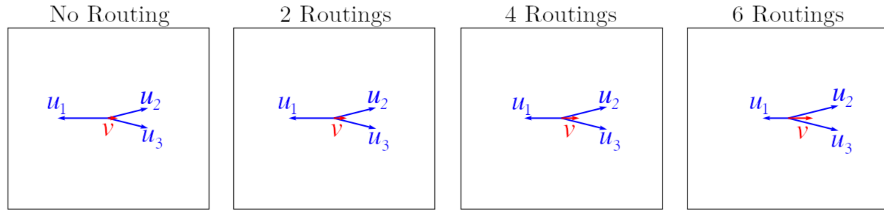

Figure 24: Additional example of the routing effects, showing that not only the longest vector has an impact. Blue arrows are the input vectors, scaled by their respective weight, red arrows are the combined result.

discussed the length of the vector corresponds to a prediction score. Effectively this results in vectors with a high prediction confidence are given more importance while less certain ones get discarded. The second effect manifests itself when $\vec{v}$ is short. This usually happens either when all contributing vectors are short, and therefore unlikely to predict the correct thing, or when there is no clear consensus between inputs, causing the contributions to cancel each other out. In both cases the small $\vec{v}$ causes the updates to the weights in Eq. 29 to be small, reducing the impact of the routing. Once again this is an advantage, since it effectively turns off the routing in cases where it would not be appropriate. However, since we only use around 3 routing steps we do not run the risk of the routing only enhancing the longest vector while discarding all others. Further, just like Fig. 23 shows that the length of the vectors can be important, Fig. 24 demonstrates that the length is irrelevant if the vector in question disagrees with the consensus.

After this lengthy treatment of the routing algorithms and its properties one question still remains: What is the practical use of this? To answer this we once again make use of a simplified example, let us assume we have two subsets of images, both contain two back-to-back top jets each, but in one subset the jets are in quadrants 1 and 3, in the other one in 2 and 4. Fig. 25 shows two of these images, with the first subset of images being shown on the left and the second one on the right. Ideally one would want to adjust the network weights to enhance the contribution coming from the sections containing the jets, while reducing the impact of the image sections that only contain noise. However, if one were to enhance the weights of quadrant 1 and 3 while reducing those of 2 and 4, this would improve classification on one subset of images, but harm the performance on the other.

Now we can consider the effect routing would have on this problem. Initially we need to convert the predictions coming from each quadrant into vectors, represented by the blue arrows that can be seen in the upper two images in Fig. 25. It seems reasonable to assume that the predictions from the jet-containing sections will, after sufficient training of the network, have a high certainty and will generally agree, while the one from the noise parts of the image will have a more or less random direction while being short. If we now perform a routing pass on the four contributions the long, more parallel vectors from the jet-section will be enhanced

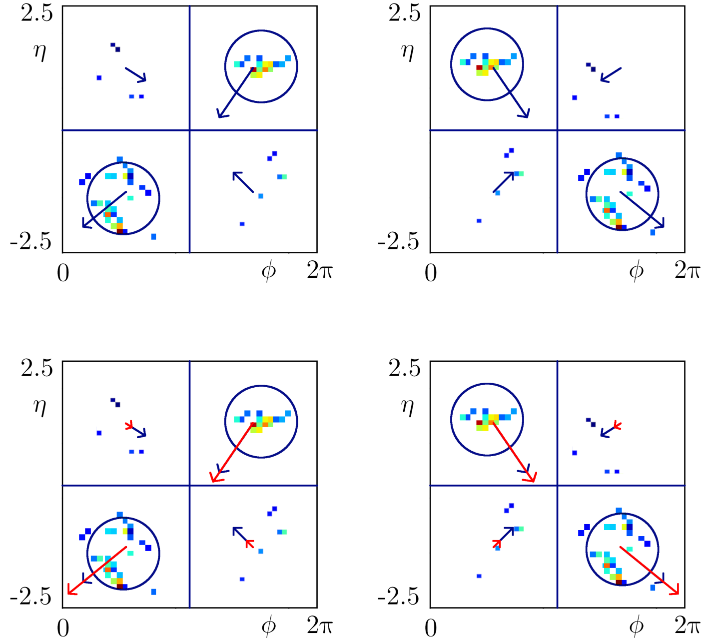

Figure 25: Example of the routing effects using a di-top event. Blue arrows are the capsule vectors before the routing, red arrows are vectors after routing.

and the noisy will be suppressed, as shown by the red arrows in the lower half of Fig. 25. This means that the routing is able to dynamically adapt to which of the two subsets it is dealing with, enhancing the classification performance for both.

Another important reason for using the routing is that it helps to train the network to associate long vectors with high prediction scores, since the reweighing of the input contribution is not based on individual vector entries, but only on their combined length. This makes the routing a vital operation, needed for the capsule network to function.

4.4 Squashing Functions

In the previous section on the routing we briefly mentioned the so called squashing function, given by

$$v_{i'}^{(j')} = \vec{v} \rightarrow \vec{v}' = \frac{\vec{v}^2}{1+\vec{v}^2}\,\hat{v}\,. \tag{30}$$

The purpose of the squashing is twofold. For one it forces the length of the instantiation vectors to be between 0 and 1, thereby ensuring the scalar products we calculate during the routing cannot get to large, which would throw of the weight recalculation. The second purpose is more

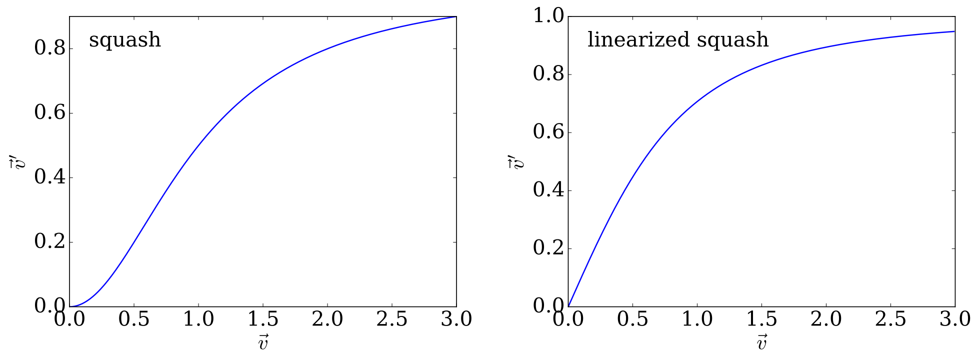

Figure 26: Left: Shape of the standard squashing function. Right: Shape of the linearized squashing.

fundamental. As previously discussed, neural networks require some source of non-linearity in order to be descriptors for non-linear problems. Normally this would be achieved through an activation function, however the two most common activation functions discussed earlier, Sigmoid and ReLU cannot be used for a capsule setup, since they are not inherently defined to accept vector inputs, and more importantly, would not work in a capsule setting. This is because both activation functions require an input space from $-\inf$ to $\inf$, however our vector length can only be positive. For Sigmoid this would mean halving its output range to $[0.5, 1)$ and ReLU would simply stop being useful, since for positive inputs it is simply the identity operation and would therefore fail to provide any non-linearity to the network. This means we a new type of activation function, and the squashing is used to fill this role.

However, when using the squashing function show in Eq. 30, which was proposed in the original paper [41], one runs into a problem. For small inputs the squashing function becomes

$$\lim_{x \to 0} \frac{\vec{v}^2}{1 + \vec{v}^2} = \frac{\vec{v}^2}{1} , \tag{31}$$

meaning these small inputs the squashing behaves like a quadratic suppression. This behavior can also be seen in the left part of Fig. 26. Such inputs appear, among other times, right after initialization, when the network just start training and all weights are still uniformly tiny. These already small values are then further suppressed by the squashing, which is applied in each capsule layer. In a deep network, in other words a network if several capsule layers, this leads to final outputs too small for the network so handle. Another problem with small inputs to the squashing function is that the gradient around zero becomes tiny. This can be very problematic for gradient descend methods, since they have a tendency to get stuck when gradients are not sufficiently large. Advanced optimizer like ADAM, described in section 3.4, incorporate momentum terms to accommodate this problem, however right after initialization these momentum terms are still zero. Both this and the quadratic suppression can cause the network to stall right after the training starts. To rectify this problem we introduce the

linearized squashing function, which essentially is just the square root of the original squashing,

$$v_{i'}^{(j')} = \vec{v} \to \vec{v}' = \frac{\vec{v}}{\sqrt{1+\vec{v}^2}}\,\hat{v}\,. \tag{32}$$

The right hand side of Fig. 26 shows the features of this function and lets us compare it to the original squashing. We can see that the general shape and domain are the same, the main difference is in the behavior around zero. Here our new squashing becomes

$$\lim_{x\to 0} \frac{\vec{v}}{\sqrt{1+\vec{v}^2}} = \frac{\vec{v}}{1}\,. \tag{33}$$

So instead of the previous quadratic suppression we now have a linear relation, hence the name linearized. Further we now also have a non-vanishing gradient round $x = 0$. This effectively solves the stalling problem during the start of the training, and allows us to build deeper capsule networks.

4.5 Margin Loss

In principle there are few requirements on our capsule network loss function, aside from the properties inherent to most loss functions, rewarding correct classifications and discouraging wrong ones. Therefore, we chose to use the margin loss proposed in [41]. The total loss L can be broken down into individual per-capsule losses.

$$L = \sum_{j'} L^{(j')}\,. \tag{34}$$

Where j' runs over all output capsules. The individual expressions are given by

$$L^{(j')} = T^{(j')} \max\left(0, m_+ - |\vec{v}^{(j')}|\right)^2 + \lambda(1 - T^{(j')}) \max\left(0, |\vec{v}^{(j')}| - m_-\right)^2\,. \tag{35}$$

With $T^{(j')}$ being the truth label for the j'-th capsule, and $\vec{v}^{(j')}$ the output of that capsule. The λ-parameter in front of the second term allows us to scale the relative contributions of the correct-capsule and wrong-capsule term. Further m_+, m_- are arbitrary parameters that describe the desired length for the correct and incorrect capsule, respectively. We chose $m_+ = 0.9$ and $m_- = 0.1$. Initially Eq. 35 can seem somewhat convoluted, so we will break it further down using a concrete example. We assume we have a two class problem, for example a signal vs. background classification. Therefore we have two output capsules, so $j' = 1, 2$, further we assume we have a signal event, meaning the label for class 1 is $T^{(1)} = 1$ and $T^{(0)} = 0$.

The loss for the first capsule $L^{(1)}$ now simplifies to

$$\begin{aligned} L^{(1)} &= T^{(1)} \max\left(0, m_+ - |\vec{v}^{(1)}|\right)^2 + \lambda(1 - T^{(1)}) \max\left(0, |\vec{v}^{(1)}| - m_-\right)^2 \\ &= \max\left(0, 0.9 - |\vec{v}^{(1)}|\right)^2 \ . \end{aligned} \tag{36}$$

Similarly the loss for the other capsule becomes

$$\begin{aligned} L^{(1)} &= T^{(1)} \max\left(0, m_+ - |\vec{v}^{(1)}|\right)^2 + \lambda(1 - T^{(1)}) \max\left(0, |\vec{v}^{(1)}| - m_-\right)^2 \\ &= \max\left(0, |\vec{v}^{(1)}| - 0.1\right)^2 \ . \end{aligned} \tag{37}$$

The key part of the loss function is the $0.9 - |\vec{v}^{(1)}|$ section, which gets smaller the closer $|\vec{v}^{(1)}|$ gets to 0.9, by squaring this term we ensure that larger differences are weighted more heavily. Finally, the max makes it such that once the length of $|\vec{v}^{(1)}|$ gets larger than 0.9 the loss stays constantly minimal. In conclusion, the loss function therefore drives the length of the correct capsule to be larger than $m_+ = 0.9$ and the length of the wrong one to be smaller than $m_- = 0.1$, which is exactly the desired behavior. Throughout this work we put larger emphasis on forcing the correct capsule the be longer, hence we chose $\lambda = 0.5$. The reason we do not choose $m_+ = 1.0$ and $m_- = 0.0$ is that for $|\vec{v}^{(j')}|$ reaching 1.0 is difficult because of the squashing. Also reaching 0.0 is also problematic since it would require every single entry to be 0, which clashes with the principle that the entries describe properties of features. Therefore, we choose values slightly off 0.0 and 1.0, arguing that the network can still encode features in a vector with length 0.1, but not in one with length 0.0.

4.6 ϕ-Padding

In the machine learning context padding generally refers to artificially increasing the size of data without adding real information. The most common occurrence is arguably in combination with convolutional layers. In order to understand the advantages of padding, it is instructive to first see what happens when no padding is used. The top third of Fig. 27 shows a convolution for a 4×4 sized input with a 3×3-kernel and a default stride length of 1. Because the kernel needs to fully cover a 3×3 section in oder for the convolutional operation to be well defined, there are only four valid kernel positions within the input image, leading to a 2×2 output. The main downside of this is that, information about the entries right at the edge of the image is lost, since the two entries at each edge only contribute two to output entries and the corners even to only one, while each of the four center entries influences the whole output. Another indication for this loss of information is, that when going from 16 entries in the input to 4 in the output, it is not generally possible to preserve all information included in the input.

A popular solution to this is shown in the middle of Fig. 27. This process is called

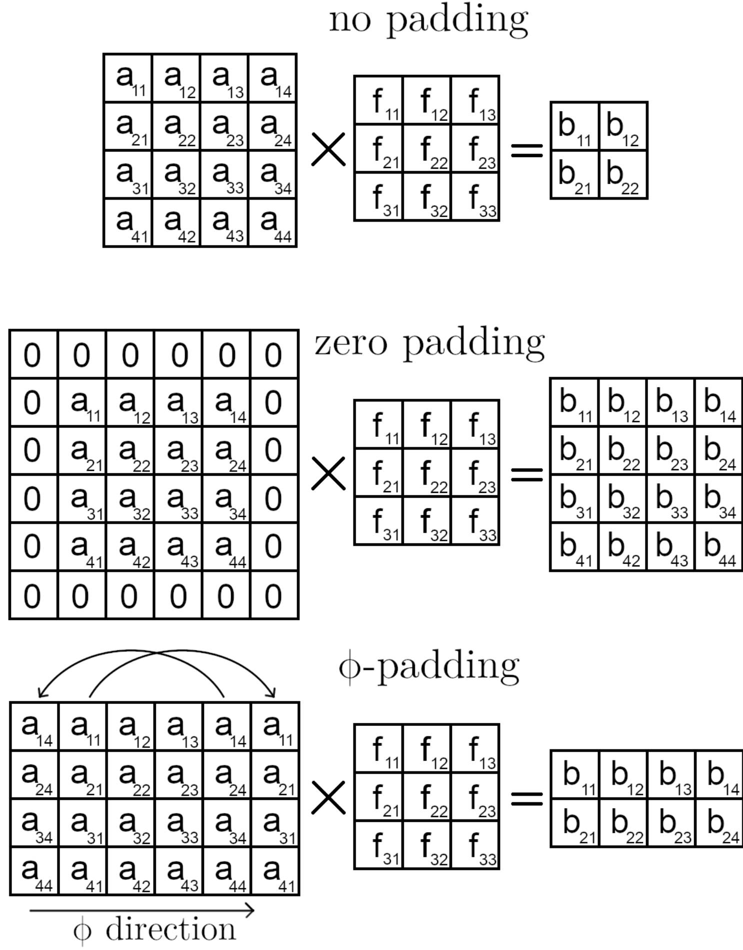

Figure 27: Illustration of different padding styles for a convolution with a 3×3 kernel. Top: no padding. Middle: zero padding. Bottom: ϕ padding

same-padding, or more generally zero-padding, and adds zeros around the input image until the output size is the same as the size of the pre-padding input. Now the kernel can extend beyond the edges of the original image without being ill-defined and the output has the same number of entries as the input. This allows the network to preserve the information about the edges.

For our capsule network approach, however, we also use a different type of padding. One of the features of our η-ϕ calorimeter images is that they should be invariant under translation in the ϕ direction. This is because ϕ corresponds to the azimuthal angle in the detector, and both the physical processes as well as the detector are symmetric under rotation along the beam axis. However, we lose this symmetry, since our 2 dimensional image is not cyclic, meaning that in the image a entire at $\phi = 1.95\pi$ and one at $\phi = 0$ will be far apart, despite being right next to each other on the detector cylinder. This becomes especially problematic when using convolutional layers, since they rely on such proximity information. One problematic example

would be a jet centered around $\phi = 0$, leading to on half of the jet getting projected onto on side of the image and the other half onto the other image side. In this case a convolution would have no way of knowing both halves are belong together.

To remedy this problem, we employ wrap-padding, here also called phi-padding when it is done only in the ϕ-direction. The principal is shown in the last third of Fig. 27, one takes the leftmost and rightmost parts of the image, copies them, and then adds them onto the respective opposite side. The number of lines padded this way is once again chosen so the input and output size in ϕ direction are equal. In our full event capsule network architectures we employ ϕ-padding before every convolution, which both helps the network deal with split jets and lets it learn the ϕ-symmetry in the events.

4.7 Choice of Classifier

In section 3.5 we discussed how ROC curves are used to evaluate the performance of a network. A question that no arises for the capsule setup is what should be used as the prediction score. For example in a standard neural network designed to separate signal from background one usually has a single output neuron with a sigmoid activation function. This results in a single output value between 0.0 and 1.0 that can directly be used as a prediction score. Similarly for multi-class CNN's or networks that use two different outputs for signal and background one usually applies a softmax function on all outputs, ensuring they add up to one. This once again allows us to use one of the outputs as our score. In the capsule case, however, this becomes less trivial. Here there is nothing to force the lengths of signal and output capsules to add up to one, and therefore, most of the time, the sum over all output capsules will be unequal to 1. One way to approach this would be to use the length of the signal capsule $|\vec{v}^{(s)}|$ as the prediction score, this does, however, come with a downside. Let us assume we have two events, for one the length of the signal capsule is 0.6 and the length of the background capsule is also 0.6, for the other event both lengths are 0.4. In both cases the network is equally confident that the event is signal, as that it is background. However if we just look at the signal capsule output and put a cutoff at 0.5 one of the events will be classified as signal and the other one as background.

A different approach would be to try and take the information contained in the background capsule into account. For example one could normalize the outputs by hand, so to speak, by calculating

$$|\vec{v}^{(s)}|_{\text{norm}} = \frac{|\vec{v}^{(s)}|}{|\vec{v}^{(s)}| + |\vec{v}^{(b)}|} , \tag{38}$$

where $|\vec{v}^{(b)}|$ is the length of the background capsule. When we are dealing with more than one

background classes this can be generalized to

$$|\vec{v}^{(s)}|_{\text{norm}} = \frac{|\vec{v}^{(s)}|}{|\vec{v}^{(s)}| + \sum_n |\vec{v}^{(b,n)}|}\,, \tag{39}$$

with $|\vec{v}^{(b,n)}|$ being length of the n-th background capsule.

Another way to incorporate the background output is by defining what we call the signal-likeness $|\vec{v}^{(s)}|_{\text{like}}$ as

$$|\vec{v}^{(s)}|_{\text{like}} = |\vec{v}^{(s)}| - \sum_n |\vec{v}^{(b,n)}|\,. \tag{40}$$

In our studies we saw the signal-likeness approach to be on average the most successful, hence we will be using it as the default throughout the remainder of this thesis. However there seems to be no method that is clearly superior in all cases, and, depending on the specific task, the choice of classifier can have a severe impact on curve shape and AUC. This indicates that the choice of classifier should be made on a case-by-case basis.

5 Whole Event Inputs

5.1 Event Generation

Throughout this thesis we use Sherpa2.2.5 [42] at leading order with the Comix [43] matrix element generator. For our parton density function we used the 5-flavor LO NNPDF3.0 PDF set [44]. We include parton shower and hadronization effects, but ignore out pile-up and underlying event.

For fast detector simulation we used Delphes 3.4.1 [45] with the default ATLAS card, adapting the jet-clustering radius and algorithm to better work for our individual processes, depending for example on whether we wanted hadronic top jets to be viewed as one fat jet or three individual jets. Specific algorithms and radii are mentioned in the respective sections. For the jet clustering we used the FastJet [46, 47] package.

After the detector simulation we use ROOT [48] to perform initial selection cuts and to extract the per event calorimeter and tracking data. We then use this data to produce a calorimeter image that covers a range of $[0, 2\pi)$ in ϕ and $[-2.5, 2.5]$ in η, separated into 180 bins in each direction, resulting in a images with 180×180 pixels. Each pixel therefore covers 0.028 in ϕ and 0.035 in η. The value of each pixel is equal to the total transversal energy deposited in the respective η-ϕ bin.

5.2 Data Storage

Now that we have generated our full events and translated them into calorimeter images we need a way to store them. The standard approach is to simply flattened the image and stored the value of each individual pixel. This works for jet images with a 40×40 resolution, since each image contains only 1600 pixels, and the images were focused on the jets, so there was less whitespace than in full events. Both of these things no longer apply for the 180×180 events. Not only do we now have 20 times more pixels, the images are also a lot more sparse, specifically out of the 32400 total pixels there are rarely more than 300 with non zero values. Since it would be wasteful to save all the zero valued pixels we decided to implement a format that only saves value and position of the non-zero pixels. To achieve this we once again flatten the image into a one dimensional array with 32400 entries. We then parse through this array and each time we hit a non-zero entry we save both its position, p_i in the one dimensional array as well as the corresponding value v_i. The index i indicates the the position and value belong to the i-th non zero entry in the image. In the end we get a list with this composition: $[p_0, v_0, p_1, v_1, ..., p_n, v_n]$. Since we are using HDF5 [49] tables for our save format we need a fixed table size and therefore a fixed n, which we choose to be $n = 400$. If we end up having

less then 400 non-zero pixels in the image we simply fill the rest of the list with zeros, and if we have more we sort the entries by p_T and drop the softest constituents.

To later restore the original image we go thorough our list of positions and values and for each position calculate the original η and ϕ positions using

$$\eta_i = p_i \bmod bins_\eta \tag{41}$$

$$\phi_i = \frac{(p_i - \eta_i)}{bins_\eta} , \tag{42}$$

where mod is the modulo operator, which returns the remainder of the division, and $bins_\eta$ is the amount of pixels in the η direction, 180 in our case. We then generate a 180×180 array initialized with zeros and increase the entry at (ϕ_i, η_i) by v_i. This procedure significantly reduces the file size and thereby also speeds up the network training process.

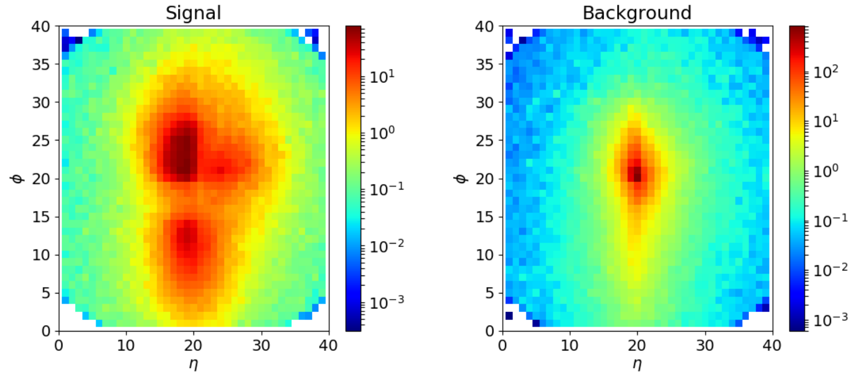

Figure 28: Overlay of preprocessed jet images. Left: Top jets signal. Right: QCD jet background. Taken from Ref. [50].

6 Top vs QCD Dataset

First we want to validate that our capsule setup is capable of replicating the results of conventional convolutional networks on a simpler, well understood sample. For this we choose the top vs QCD challenge data set [21, 26]. This set consists of a total of 2 million QCD and top jets, 60% of which are used for training, and the remaining 40% are split evenly into validation and evaluation samples. In the original data set each jet is described by a list of the 4-momenta of its 200 most energetic constituents, although most of the time the jet will have less then 200 constituents, in this case the remaining entries of the 4-momenta list will be filled with zeros.

Our main benchmark for comparison here is the Rutgers Deep Top image based convolutional network shown in Fig. 29 and described in [22]. For this network the constituent list needs to be preprocessed and translated into images. The first step consists projecting the jet onto the η-ϕ-plane, then finding the p_T-weighted centroid of the jet and shifting it to the center of the image. Following this the jet is rotated until its central axis is vertical, then the image is horizontally and vertically flipped, until the top left corner of the image contains the larges amount of total p_T. Finally the image is pixelated into 40×40 pixels and normalized so the total pixel entries sum up to 1.

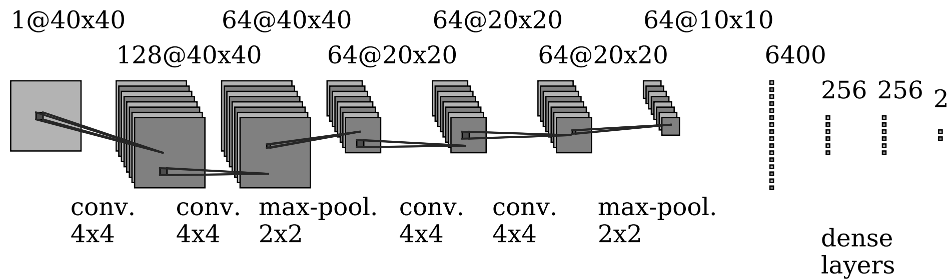

Figure 29: Rutgers Deep Top convolutional neural network [22] for top tagging.

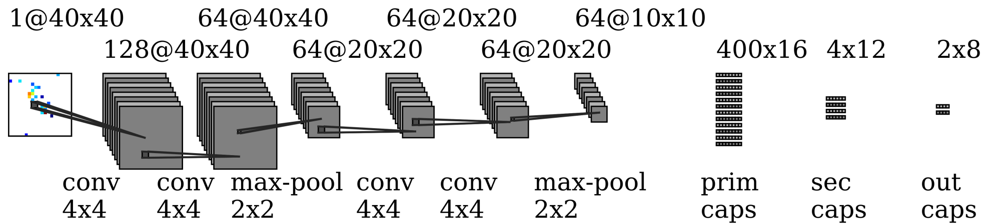

Figure 30: Capsule network designed for the top vs QCD reference data set. Convolutional structure inspired by the Rutgers Deep Top network [22]. Taken from Ref. [25].

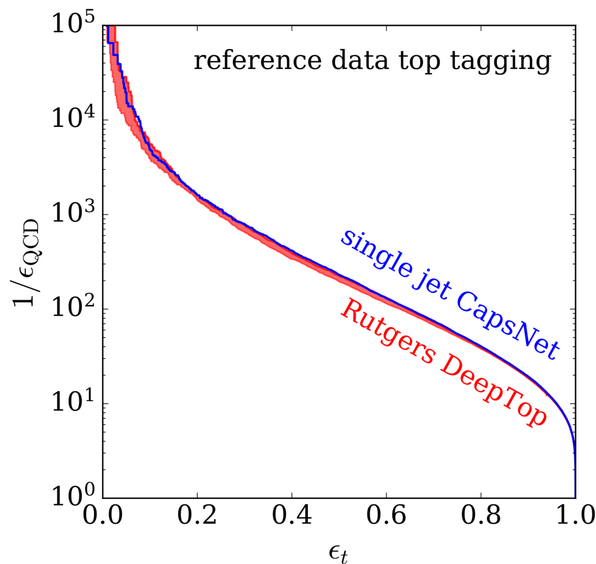

Figure 31: Comparison between the capsule network in Fig. 30 and the Rutgers Deep Top network [22] in Fig. 29 using the top vs QCD reference dataset. As shown in Ref. [25].

Fig. 28 shows an overlay of several jet images after preprocessing, as one can see there are visual differences between signal an background, most notably the three-prong nature of the hadronically decaying top jet. For the sake of comparison we chose to use the same images with identical preprocessing for both networks. Further, we model the convolutional structure of the capsule network to be quite similar to the Rutgers network, since it has proven to be very effective for top vs. QCD discrimination. The final capsule network architecture we used can be seen in Fig. 30. For the implementation of this network we used Keras [51] with a Tensorflow back-end [52]. The capsule layers are is based an implementation [40] of the original capsule paper [41]. For training we used the Adam optimizer [38] and learning rate decay with a decay factor of 0.9. Further we employed early stopping to end the training process once the validation accuracy started to drop. This base setup was also used for the other capsule network architectures described in this thesis.

The performance comparison between the two networks can be seen in Fig. 31, which shows that the capsule network is very much competitive with the Rutgers network, when dealing with a problem where there is no additional structure for the capsules to extract and take advantage of. Further, the regular preprocessing for this dataset essentially replaces the advantage of the capsule network to extract features independent of their location.

7 Heavy Mediator Decay

One of the main goals of machine learning in particle physics is to help discover new physics events that classical analysis methods might have missed. Many of these potential new physics processes involve a heavy neutral mediator, that could, among other things, be a connection between the standard model and dark matter [53]. For their similarity to the standard model Z boson these mediators are often referred to as a Z'. In order to introduce this new heavy mediator we expand the standard model Lagrangian by

$$\mathcal{L}_{Z'} = c_{tot} \sum_q c_{qq} (\bar{q} \gamma_\mu q Z'^\mu) \; . \tag{43}$$

Here Z' is our heavy mediator that couples to all quarks. The constants c_{tot} and c_{qq} describe the over all coupling and the coupling strengths to the individual quarks. We implement this model using FeynRules [54, 55], allowing us to generate a UFO file [56] we can use in Sherpa. The model parameters we used in this case were $c_{tot} = 1$, $c_{qq} = 1$, $M_{Z'} = 1$TeV and $\Gamma_{Z'} = 1$GeV. Our results are largely independent of the specific values of the parameters, only a significantly larger width could potentially have an impact, since it would cause us to no longer have a sharp resonance peak. Moreover, since we use an even split between signal and background events in our datasets, the cross section of this process is not relevant for us.

The process we want to look at in detail is the decay of the heavy Z' mediator into two tops,

$$pp \to Z' \to t\bar{t} \; . \tag{44}$$

Focusing only on decays into top-jets is well motivated in practice, for one it allows us to use dedicated top-taggers to reject all light jet events, which already cuts out a huge part of the background. In addition, it also lets us draw from the experiences and results made in other machine learing projects, many of which focused on top-tagging.

7.1 QCD Background

As our first step we want to see whether our capsule network can function as a effective di-top tagger by comparing it to the Rutgers top-tagger from section 6 in Fig 29. This means the first background we will be investigating are two light quarks jets produced via QCD processes:

$$pp \to jj \qquad \text{with } j = g, u, d, c, s, b \; . \tag{45}$$

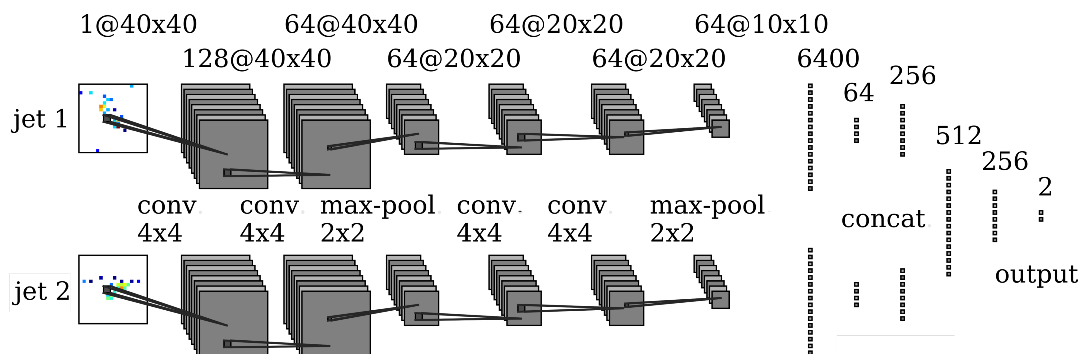

Figure 32: Rutgers Deep Top network, modified to accept two jet images as input, to allow for di-top tagging. Taken from Ref. [25].

We demand for both signal and background events to have at least two jets with

$$p_{T,j} > 350 \text{ GeV} \qquad \text{and} \qquad |\eta_j| < 2.0 \; . \tag{46}$$

This rejects the low p_T jet background and ensures the jets are located well within our image range of $\eta \in [-2.5, 2.5]$. In total we produced $500,000$ events after cuts, with a $1:1$ signal to background ratio. We split these into $300,000$ events for our training sample, and $100,000$ for validation and evaluation each.

Now we need to modify both the capsule network and our benchmark, the Rutgers Deep Top network, to accept an input of two jet images. To this end we modify the Rutgers network by duplicating the convolutional structure, essentially running two parallel convolutional streams which we then combine in the dense section, see Fig. 32, also in comparison to Fig. 29. To generate the input images we select the two most energetic jets from the event file and store them in a way identical to how it is done for the top vs QCD challenge data. From this we then generate two separate 40×40 jet images, preprocess them independently, and finally feed one jet image to each of the two convolutional streams.

For the capsule network these modifications are more involved, since on the one hand we want to use the full 180×180 image format as our input, but on the other hand cannot include event level information without invalidating the comparison. Therefore, we mask the event level features in our images. This process is depicted in Fig. 33. First we translate the events into images as described in section 5 and pad those images by 20 pixels using wrap-padding in ϕ-direction and using zero-padding in η-direction (first panel of Fig. 33). We than select the two most energetic jets in the image, and cut out two 40×40 pixel regions around the centers of the jets (second and third panel). The previous padding ensures these sections do not extend past the image borders. These sections are then pasted into predefined positions in an otherwise empty 180×180 image (last panel in Fig. 33). This process removes the event level information from the input images, while leaving sub-jet features intact,

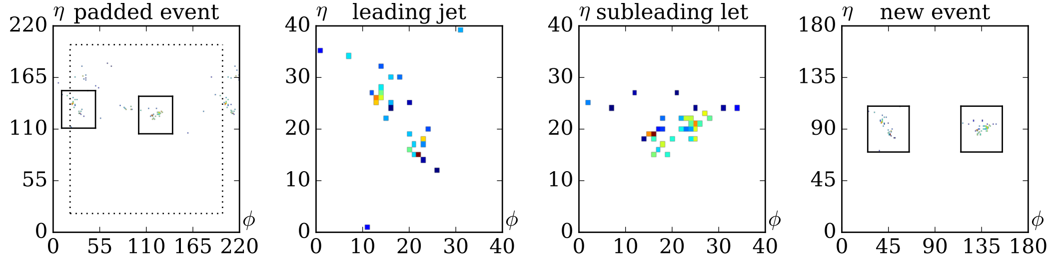

Figure 33: Processing of the event images to a pair of top jets, in order to obscure event level information. Taken from Ref. [25].

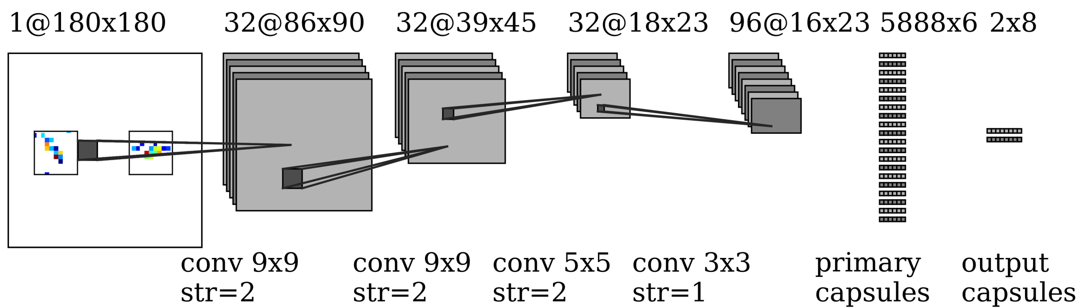

Figure 34: Capsule network 'conv'-architecture, designed for di-top tagging with 180 images. Also shown in Ref. [25].

These 180×180 images are then passed to our capsule network. Fig. 34 shows our first full resolution capsule network which was used for this problem. The first section contains four convolutions with progressively smaller kernel sizes, starting with 9×9 and ending at 3×3. Not shown in the figure are the ϕ-padding layers before each convolution, the padding amount is always equal to half the size of the following convolutional kernel, rounded down. Additionally, we employ strided convolutions, allowing us to reduce the size of the image, and with it the computational resources required to run our network. We use one routing layer, since during our experiments, architectures with additional routing layers showed no performance improvement.

In Fig. 35 we can see one ROC curve for the Rutgers double top tagger in red, and two for the capsule network in blue. The two capsule curves represent different classifiers. The one with the better background rejection in the low signal efficiency region was generated using $|\vec{v}^{(s)}|$ as the classifier, while for the other one we used $|\vec{v}^{(s)}|_{\text{like}}$ defined in Eq. 40. The first thing we can notice is that the signal-likeness actually decreases the performance in the low signal efficiency region, which for our purposes tends to be the most relevant. This is the only case we have encountered so far, for which using the pure signal capsule length $|\vec{v}^{(s)}|$ is advantageous compared to the signal-likeness. This demonstrates that, as said in section 4.7, choice of classifiers seems to be problem dependent. When we compare the $|\vec{v}^{(s)}|$ capsule

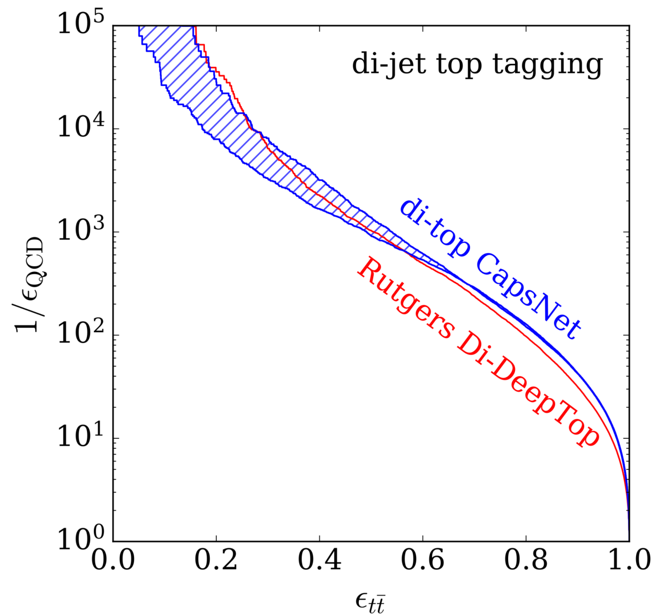

Figure 35: Comparison of di-top tagging performance between the capsule network and the modified Rutgers Deep Top network. Taken from Ref. [25].

curve to the Rutgers curve we see that the capsules perform better in the high signal efficiency region, and match the Rutgers results in the low efficiency area. This shows that the capsule setup is indeed competitive compared to other machine learning methods.

7.2 $t\bar{t}$ Background

In the previous section we saw that the capsules are able to match standard convolutional neural networks in extracting sub-jet features. Now we want to see if the capsule setup can be used to learn event level information. Mirroring the previous section, we will initially test how well the network performs when we remove the sub-jet information. However, it is next to impossible to remove jet features from an image while preserving the event information. What we do instead is choose a classification task in which the jet features are nearly identical for signal and background and therefore irrelevant for the classification task. To this end we keep the same heavy mediator decay as the signal replace the background with QCD mediated di-top production:

$$pp \to t\bar{t} \, . \tag{47}$$

With this we now have top pairs, leading to top-jets, in both signal and background, however the kinematics of the processes are vastly different. The best example for this is the invariant

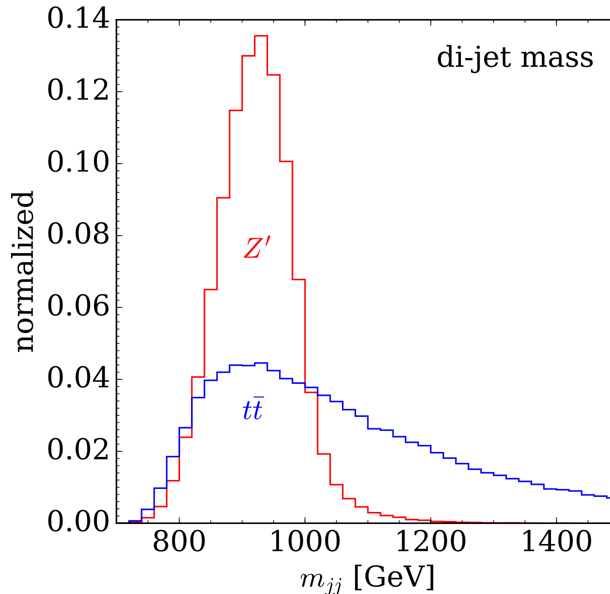

Figure 36: Invariant di-jet mass of the two leading jets for a heavy mediator decay signal and a continuum $t\bar{t}$ background. Taken from Ref. [25].

di-jet mass shown in Fig. 36. There is a clear peak around the mass of the Z' mediator for the red signal curve, and a continuous mass spectrum for the blue background curve, which is exactly as one would expect from a heavy decay process and a QCD process, respectively.

In this case the previous benchmark top tagger should fail, as there are genuine top jets in both signal and background. Instead we employ a different, classical, analysis method that also relies on event level information; a Boosted Decision Tree (BDT). In order to train the BDT we used the pseudo-rapidity of the leading jet (j_1) and the sub-leading jet (j_2), η_{j_1} and η_{j_2}, their transverse momentum, p_{Tj_1} and p_{Tj_2} and the invariant di-jet mass m_{jj}. From Fig. 36 we see that the invariant mass is very well suited to separate background from signal. The BDT itself was implemented using AdaBoost [1] from the SciKit-learn Python package [57]. The tree had 100 estimators and a maximal depth of 3, the training data we used for the BDT was identical to the one used to train the capsule networks.

The 'conv' network shown in Fig. 34 worked well on the $Z' \to t\bar{t}$ vs. QCD $\to$ light jets samples, but loses to the BDT for the $Z' \to t\bar{t}$ vs. QCD $\to t\bar{t}$ case, as the 'BDT' and 'CapsNet conv' curves in Fig. 37 show. The biggest difference between the two samples is what features are relevant for classification, for the light jet events sub-jet features are the main handle used to distinguish signal from background, while for the $t\bar{t}$ background version the network needs to learn event level features. What sets these two types of features apart is their relative size. The averages top jet at the p_T we are looking at will cover roughly a 40×40 section of

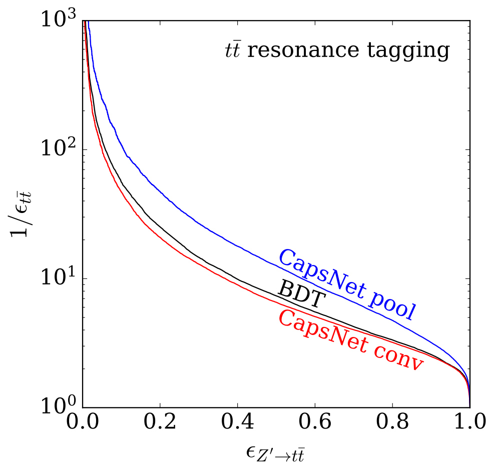

Figure 37: Comparison of $t\bar{t}$ tagging performance between the two capsule network architectures and the BDT. Taken from Ref. [25].

the image, meaning that all relevant sub-jet features are constrained within that range, on the other hand, event level features are present in the full 180×180 image. The next thing we need to take into account is the size of our convolutional kernels. As can bee seen in Fig 34 the size of our initial kernel is rather small compared to the jet size (9×9 kernel) which is appropriate for looking inside jets and learning their substructure. However when trying to extract event information the network should ideally be able to see the whole jet as one entity and be able to determine its position and energy in relation to other jets. This is where the small kernel size become as problem, since the network is forced to reconstruct the jet from a series of small jet-subsections. This is possible, after all Fig. 37 shows the network has classification power, however we can restructure the convolutional part of out network to try and make this task easier.

Therefore, in order extract the large scale structures and outperform the BDT we can increase the relative kernel size, meaning using either a small image or bigger kernel. We chose to do both, which leads us to the 'pooling architecture' depicted in Fig. 38. In the first step we using pooling operations to reduce the image size down to one sixteenth of the original resolution. Here we also crate two parallel network streams, one using max pooling, the other one average pooling. The idea behind this is that max pooling will preserve the energy values of the most energetic constituents, while the average pooling preserves the total jet energy, by doing both in parallel we are able to keep both. The max pooling stream is then passed to a convolutional layer with a large kernel, see Fig. 38, the purpose of which is to find large structures, like whole jets. This is followed by another max-pooling layer and a final convolution, before reshaping the output into capsules. The average-pooling stream is again split into another three sub-streams, the first of which gets passed to a convolutional setup similar to the max-pooling stream, the other two are average-pooled again, down to an image

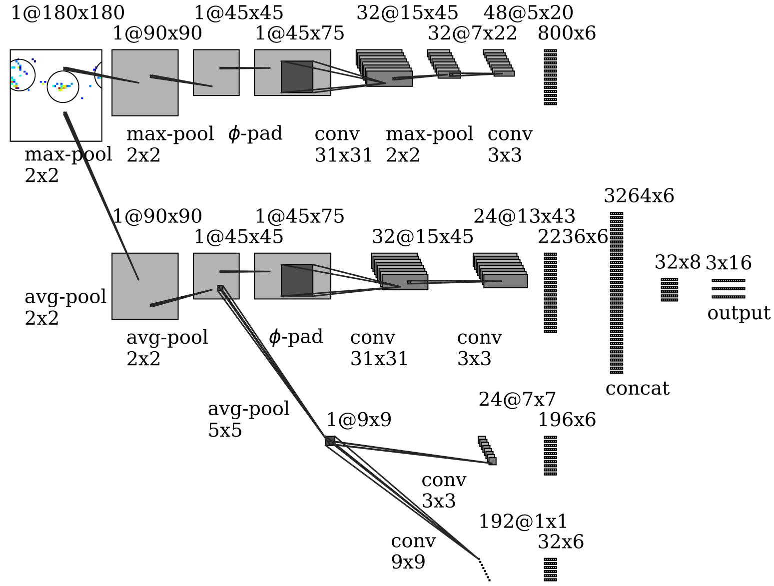

Figure 38: 'Pooling' capsule network architecture, specialized on extracting event level kinematics. Taken from Ref. [25].

size of 9×9 pixels. These small images are then fed to two different convolutional layers. One with a comparatively small kernel, intended to learn relations between jets, and one with a kernel covering the entire image, designed to give the network a handle on global observables, such as total event energy. These last two small convolutions, unlike all other convolutions in the network, do not have a *phi*-padding layer, in order to preserve the total energy amount of in each event.

This specialized network now not only surpasses the previous 'conv' architecture, but also gives a significant improvement in performance compared to the BDT, as can be seen from 'CapsNet pool' curve in Fig. 37. Proving that the capsule approach is capable of outperforming classical analysis methods.

7.3 Combined Backgrounds

Combining the results from the previous sections we are capable of distinguishing $Z' \to t\bar{t}$ from both a $t\bar{t}$ and a QCD background. This begs the question whether we can somehow combine the two. In an actual use-case one generally has more then one process contributing to the background. Ideally, our event tagger would not just be able to differentiate between signal and one background type, but all backgrounds. Or, even better, actually classify the background processes themselves.

	Binary label	Multi label
Signal	Class 1 50%	Class 2 33.3%
$Bkg_{t\bar{t}}$	Class 0 25%	Class 1 33.3%
Bkg_{QCD}	Class 0 25%	Class 0 33.3%

Table 1: Table showing the mixing ratios of signal and backgrounds for two and three label cases.

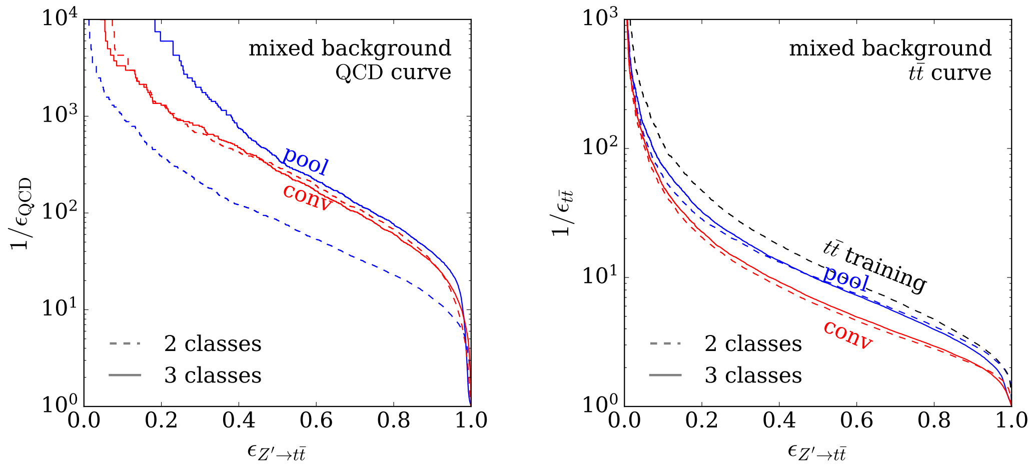

Figure 39: Left: Comparison of QCD rejection power for 'pooling' and 'conv' architecture and for two and three output classes. Right: Same comparison for $t\bar{t}$ rejection. The black $t\bar{t}$ line corresponds to the 'pooling' curve from Fig. 37. Taken from Ref. [25].

In an effort to explore how well the capsule network can tackle this problem we now move two a mixed background sample. As previously indicated we have two approaches, for one we can designate the $Z' \to t\bar{t}$ events as the signal class and put both the $t\bar{t}$ and a QCD background into one, general, background class, or we can, once again, have $Z' \to t\bar{t}$ as the signal class, but put the $t\bar{t}$ background and QCD background into two individual classes. We try both approaches, each time ensuring our samples contain equal amount of each class. This is further illustrated by table 1.

We test both the 'conv' as well as the 'pooling' architecture. To accommodate the varying number of classes in our samples we need two versions of each network, one with two output capsules, the other one with three. In order to investigate the performance of the setups we have to split the samples back into signal vs. $t\bar{t}$ and signal vs. QCD. All networks were still, however, trained on the mixed samples. Before we start discussing the individual results we should first mention theoretical differences between the 2- and 3-class cases. As mentioned in section 4 the individual entries of output vectors of the signal and background encode features

of signal events and background events respectively. This means that in the 2-class case the common features of both background types need to be combined into one capsule, while in the 3-class case the features of each class can be put into individual output capsules. The effects of this differ between setup and background type.

The results are shown in Fig. 39, we will start by discussing the QCD background. Here we can see that for the 'conv' architecture the difference between two and three classes is barely noticeable. This is because the 'conv' setup can easily extract sub-jet features thanks to its high resolution, but is less well suited for extraction of event-level information. This means that the capsule for the combined background of the 2-class setup will encode almost excursively sub-jet features. If one instead uses a dedicated QCD capsule, the features of this QCD capsule would therefore be very similar to the one of the combined background capsule, meaning that going from two to three output classes offers little to no improvement here. This is different for the 'pooling' setup, which clearly benefits from the additional output class. The main reason for this is that the 'pooling' network, by design, is able to learn event level features very well. Therefore, unlike in the 'conv' version, a combined background capsule will mainly contain event level information, resulting in a comparatively weak QCD rejection. However when we now go from two to three classes the network can have a dedicated $t\bar{t}$ capsule with event features, and another QCD capsule for jet features. This results in the huge performance discrepancy between the two curves we observe.

In the $t\bar{t}$ background plot, depicted on the right in Fig. 39, we once again see that the 'pooling' architecture performs significantly better. We do, however, lose some classification power by going from a network trained on pure $t\bar{t}$ background to one trained on a mixed background, which was to be expected since the network has to allocate computational power to the additional background type. When looking at the 2- and 3-class curves we see little difference, with the 'pooling' network this is because in the two class case the network looks for event level features in both the QCD and $t\bar{t}$ background. Since both are QCD mediated di-jet events they have rather similar kinematics and therefore similar event features. This means that the combined background capsule will mostly encode the same event level features that a dedicated $t\bar{t}$ capsule would encode, meaning the additional output class offers little improvement. We observe a similar effect with the 'conv' architecture, although the cause is the opposite. This time the network is inherently bad at learning event features, allowing the few event level features that the network does end up learning to be easily combined with the QCD sub-jet features into one mixed background capsule. This means the additional class has little effect.

In conclusion we see that we the 'pooling' capsule network is able to handle multiple backgrounds and is not only capable of separating the individual backgrounds, but also gets a performance improvement from this additional degree of separation.

Correlations of capsule entries Background Events Only

	p_{Tj_1}	p_{Tj_2}	m_{j_1}	m_{j_2}	m_{jj}	η_{j_1}	η_{j_2}	ϕ_{j_1}	ϕ_{j_2}
bkg cap entry 1	-0.44	-0.42	-0.01	-0.00	-0.34	-0.00	0.02	0.05	-0.04
bkg cap entry 2	0.64	0.58	0.13	0.08	0.51	-0.03	-0.03	-0.03	0.02
bkg cap entry 3	-0.20	-0.19	-0.02	-0.01	-0.20	-0.01	-0.03	-0.03	0.11
bkg cap entry 4	0.40	0.38	0.01	-0.01	0.26	0.13	0.11	0.06	-0.02
bkg cap entry 5	0.28	0.25	0.07	0.06	0.23	0.08	0.06	-0.00	0.03
bkg cap entry 6	0.45	0.40	0.11	0.07	0.35	0.03	0.02	0.02	0.03
bkg cap entry 7	0.08	0.08	-0.01	-0.00	0.12	0.08	0.08	-0.01	-0.04
bkg cap entry 8	0.39	0.36	0.02	0.00	0.25	0.01	0.01	-0.04	0.07

Correlations of capsule entries Signal Events Only

	p_{Tj_1}	p_{Tj_2}	m_{j_1}	m_{j_2}	m_{jj}	η_{j_1}	η_{j_2}	ϕ_{j_1}	ϕ_{j_2}
signal cap entry 1	-0.05	-0.08	-0.03	-0.07	-0.09	-0.79	-0.78	0.04	0.04
signal cap entry 2	0.04	0.05	0.11	0.11	0.11	0.12	0.11	-0.06	-0.06
signal cap entry 3	0.14	0.14	0.09	0.08	0.10	-0.22	-0.23	-0.07	-0.07
signal cap entry 4	0.04	0.04	-0.04	-0.08	-0.28	0.25	0.24	0.02	0.02
signal cap entry 5	-0.08	-0.08	-0.09	-0.10	-0.11	-0.31	-0.31	0.03	0.01
signal cap entry 6	-0.01	0.00	-0.13	-0.14	-0.09	0.13	0.15	0.06	0.03
signal cap entry 7	-0.01	-0.03	0.04	0.01	0.01	-0.76	-0.76	0.00	-0.00
signal cap entry 8	0.00	-0.02	-0.02	-0.07	-0.20	-0.69	-0.70	0.03	0.01

Figure 40: Left: Correlation table for signal capsule entries and physical observable of signal events. Right: Same table for background entries and events.

7.4 Understanding Capsule Entries

One major field of interest in machine learning is figuring out what exactly the networks learn, both to learn new and interesting observables used by the network that can have potential applications for other analyses and to make the neural network approach less black-box-esque. Ideally, the fact that in a capsule network setup capsule entries correspond to features should make this easier. To investigate this further we take our 'conv' architecture shown in Fig. 34 and look into the output capsule layer of the network. Here we have two capsules, one attributed to signal, one for background. For each event we classify, both capsules will output one vector each. Normally we would then calculate the Euclidean length of those two vectors, to obtain a signal prediction score and a background prediction score. However for our purposes we now skip the length calculation and instead look at the entries in the two output vectors. in order to learn if these entries correspond to physical variables we can understand, we calculate the correlations between these capsule entries and a list of general physics observables, such as p_T, mass, ϕ and η of the jets and the di-jet mass. Fig. 40 shows the table with these correlations, separated into the response of the background capsule to background events (left) and the response of the signal capsule to signal events (right). We find that there are some correlations between the background capsule entries and the jet p_T's as well as the di-jet mass. For the signal capsule we see correlations to the jet η position. However, these correlations vary in strength and even switch between correlation and anti-correlation. So while we can determine that the capsules include useful information, it is not entirely clear how.

Further insight was however by our Hamburg collaborators Gregor Kasieczka and Hermann Frost, who performed an in-depth study on how the capsules encode information. They used a simplified version of the capsule network, the output layer of which had two capsules

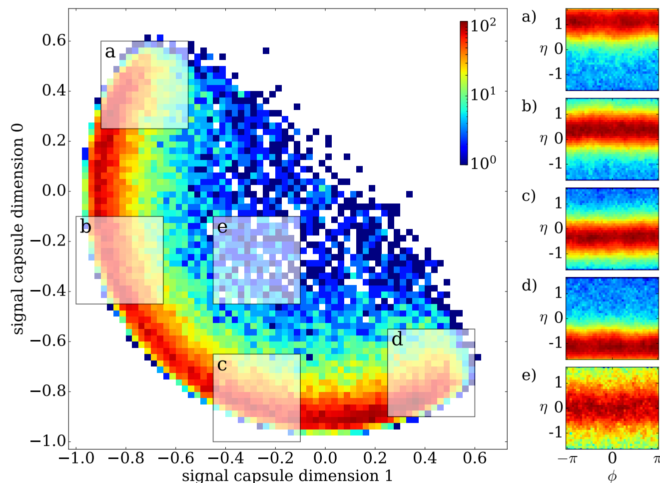

Figure 41: Main image: Overlay of signal capsule output vectors for signal events. Small images on the right: Stacked input calorimeter images. Images made by Hermann Frost and taken from Ref. [25]

with dimension 2. This meant the prediction vectors were also two dimensional and could therefore be plotted as an image. Fig. 41 and Fig. 42 plot these prediction vectors, starting from the origin, the for the signal and background capsule respectively. The first prominent feature of both plots is that the vectors from a partial circle with a radius of approximately 1 around the origin. For one, this is not unexpected, since the length of the signal prediction vector should be around 1 for signal events, and similarly for background events and the background capsule vector. However this also mean that the length of the prediction vectors are effectively fixed, leaving only the angle to encode information. In order to extract what this information is, the our colleagues selected individual sections along the circles, took at the original event images that belonged to these sections and overlayed them. These are the five small images in Fig. 41 and 42. From these we can deduce that, for the signal capsule, the angle describes the η position of both jets. Further, it indicated a event is likely to be classified as signal if both jets have the same η. This makes sense when we think about the kinematic of our signal process. After the Z' is it decays into two top jets, these jets will be back-to-back in the rest frame of the Z'. However if the Z' itself was boosted along the beam direction, both of the jets will also be equally boosted in the lab-frame, causing them to have the same η. Fig. 42 shows the same for the background capsule, where we can see that an event is more background like when the two jets are back-to-back in the lab-frame. This usually is the case for QCD-mediate di-jet events.

This leads us to a key point in understanding capsule outputs. The capsules cannot directly encode information in their entries, since the total length of the vectors is used as a prediction score and therefore fixed. They can, however, indirectly represent this information through the polar angle of the vector.

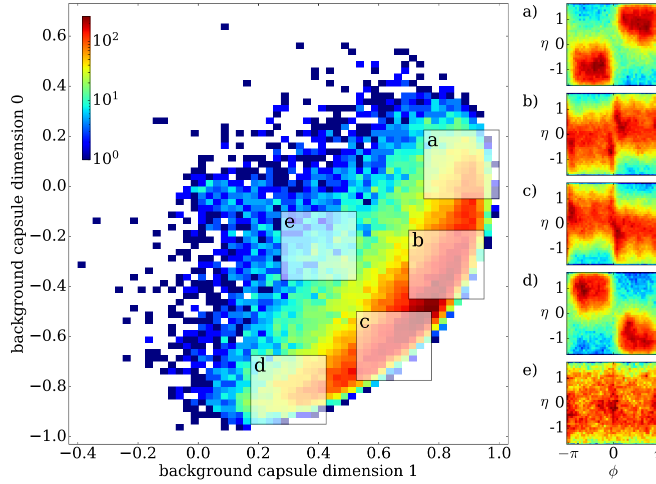

Figure 42: Main image: Overlay of background capsule output vectors for background events. Small images on the right: Stacked input calorimeter images. Images made by Hermann Frost and taken from Ref. [25]

We can now apply this principle to our higher dimensional capsule outputs. The n-dimensional vectors with entries $x_1, ...x_n$ can be expressed as one radius r and $n-1$ angles $\xi_1, ...\xi_n$ [58]. We now can take the relations

$$\begin{aligned} x_1 &= r\cos(\xi_1) \\ x_2 &= r\sin(\xi_1)\cos(\xi_2) \\ &\vdots \\ x_{n-1} &= r\sin(\xi_1)...\sin(\xi_{n-2})\cos(\xi_{n-1}) \\ x_n &= r\sin(\xi_1)...\sin(\xi_{n-2})\sin(\xi_{n-1})\ , \end{aligned} \tag{48}$$

which can be inverted and rewritten as

$$\begin{aligned} \xi_1 &= \arccos(\frac{x_1}{\sqrt{x_n^2 + x_{n-1}^2 + ... + x_1^2}}) \\ \xi_2 &= \arccos(\frac{x_2}{\sqrt{x_n^2 + x_{n-1}^2 + ... + x_2^2}}) \\ &\vdots \\ \xi_{n-2} &= \arccos(\frac{x_{n-2}}{\sqrt{x_n^2 + x_{n-1}^2 + x_{n-2}^2}}) \\ \xi_{n-1} &= \arccos(\frac{x_{n-1}}{\sqrt{x_n^2 + x_{n-1}^2}}) \\ r &= \sqrt{x_1^2 + x_2^2 + ... + x_{n-1}^2 + x_n^2}\ . \end{aligned} \tag{49}$$

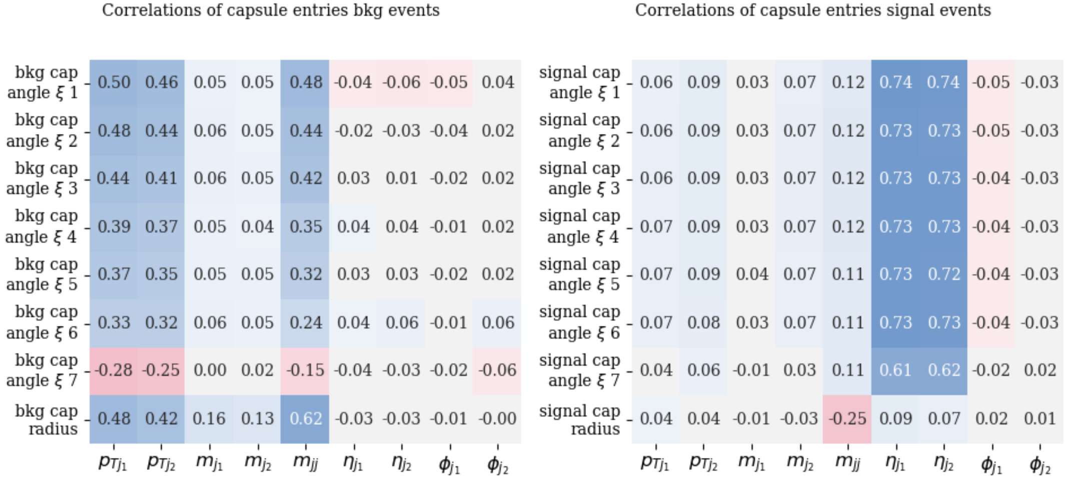

Figure 43: Left: Correlation table for signal capsule entries translated into polar angles and physical observable of signal events. Right: Same table for background entries and events.

Using these relations we can translate our 8-dimensional capsule vector into one radius and seven angles, and then calculate the correlations to physical observables. The results are once again split into signal and background capsules, and shown in Fig. 43. The angles show correlation to the same variables as the absolute entries, the background angels are once again correlated to the p_T of both jets and the di-jet mass, while the signal angles have correlations to the jet η positions. However, when using the angles these correlations are clearer and more evenly distributed when compared to Fig. 40. Further, we see that the correlation tables agree with the Hamburg results from Fig. 41, as both show that the η of the jets is important for signal classification. We also gain additional insight into the background classification, by seeing that the di-jet mass is a deciding factor for how background like a process is, something that was not obvious for Fig. 42 alone. Lastly, the fact that the correlations are shared between all angles indicates that each angle does not encode one sole observable, but rather that all angles represent linear combinations of several different observables.

In conclusion, we see that the capsule structure indeed proves helpful when trying to understand what the network uses for classification.

Figure 44: Left: signal event, associated di-top Higgs production. Right: $t\bar{t}b\bar{b}$ background.

8 $t\bar{t}H$

We have seen that the capsule network is very much capable of separating both top from QCD jets as well as identify heavy mediator decay events. Now it is time to move on to more complex signatures, specifically one of the more complex LHC processes, associated top-Higgs production. Fig. 44 shows the signal process and the main background. What makes tagging these events so difficult is for one the small cross-section of the signal event combined with the comparatively high background cross-section, and additionally the complexity of the final state, which can include up to eight separate jets, not even taking initial- or final state radiation into account. For our proposes, we only take Higgs decay into two b's into account, which is reasonable considering the high $H \to b\bar{b}$ branching ratio. We further restrict ourselves to semi-leptonic top decays for triggering.

The event generation largely follows the work flow outlined in section 5.1, the only significant difference being that this time we cluster the jets using an anti-k_T algorithm with a R $= 0.4$ radius and tag the jets as b-jets if there is a generator level b parton within $\delta R \leq 0.4$ of the jet center. For each event we save p_T, η, ϕ, m, b-tags and the PDG-ID of the b partons parent particle of the ten most energetic jets, as well as kinetic information about the lepton and the missing transverse energy. Further, in order to meet our previous requirements we enforce the following decay processes $H \to b\bar{b}$, $t \to bl^+\nu_l$ and $\bar{t} \to \bar{b}jj$ with $j = u, d, c, s$ and $l = e, \mu$, our reason for differentiating between top and anti-top decays is to ensure the semi-leptonic top decay.

In order for our results to be comparable to other analyses we implement a series of cuts:

1. exactly one lepton with $p_{Tl} \geq 5$GeV and $|\eta_l| \leq 2.5$;
2. 6 or more jets (anti-K_T, R $= 0.4$) with $p_{Tj} > 20$ GeV and $|\eta_l| \leq 2.3$;
3. exactly 4 b-tagged jets

This time we generated a combined total of $400,000$ signal and background events after cuts, which were split into 3 parts for training an 1 part each for validation and evaluation.

In principle, the essentially perfect b-tagging could have some influence on our results. However, since both signal and background have four true b-jets, the difference caused by the

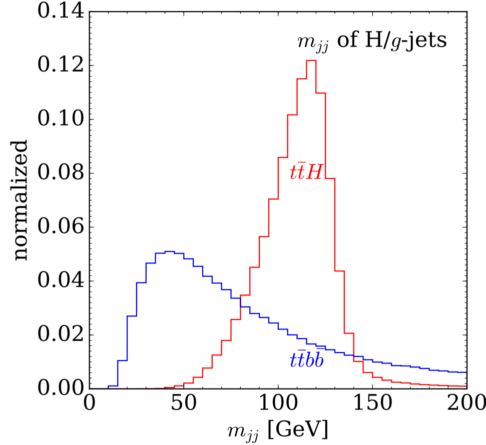

Figure 45: Invariant mass of non top-decay b-jets for signal and background. Only possible using truth level information. Taken from Ref. [25].

tagging should be the same between signal and background, and therefore have no impact on our findings.

As an additional cut we reconstruct the hadronic top jets, j_t, by taking combinations of one b-jet and two light jets and trying to minimize $|m(j_b + j_1 + j_2)| - m_t$. We demand that the difference between our jet candidate mass, m_{j_t}, and the top mass, m_t, is $|m_{j_t} - m_t| < 30\text{GeV}$. Using this reconstructed jet, we reject all with insufficient transverse momentum, specifically we require $p_{T j_t} > 200$ GeV, since we know [59] that this boosted region is of most relevance for us.

The greatest difficulty in separating $t\bar{t}H$ events from background is the reconstruction of the Higgs [60]. Specifically, if we can identify which two b-jets originate from the Higgs decay, (or alternatively, which two come from top decays) we can calculate the invariant mass of these two candidate Higgs jets. As Fig. 45 shows, this di-jet mass would be a continuum for the background, while it would form a peak around the Higgs-mass for the signal. However so far we have no way of reliably solving the combinatoric problem of which jets belong to the Higgs decay, the only we were able to make Fig. 45 is by utilizing unphysical truth information saved during the event generation.

One attempt of solving the $t\bar{t}H$ combinatorics is to look for the two b-jets closest to each other. Since the Higgs will usually be boosted in the regime we are interested in, the b-jets resulting from the decay will be very collimated. Fig. 46 shows both the smallest distance in the η-ϕ plane between two b-jets (left) as well as the di-jet mass of these closest jets. However, as we can see, there is still a significant overlap between signal and background, demonstrating that this is far from a perfect solution.

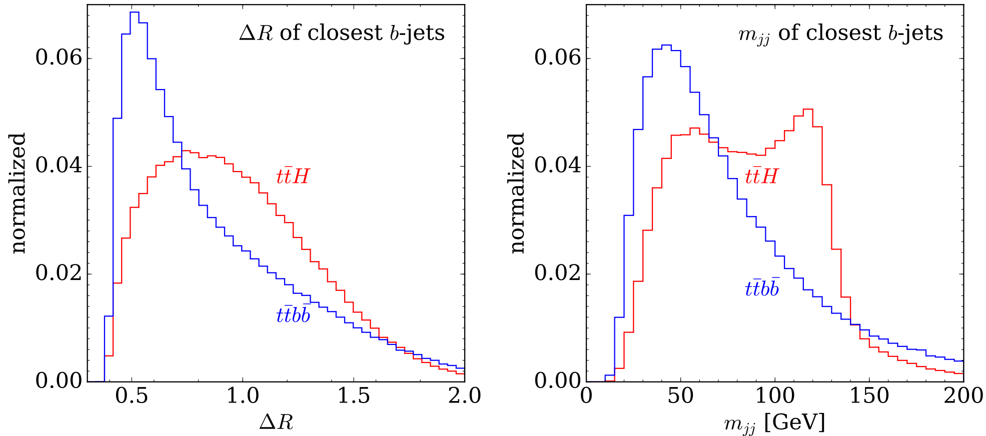

Figure 46: Left: Smallest distance between two b-jets in signal (red) and background (blue). Right: Invariant di-jet mass of closest b-jets. Taken from Ref. [25].

These combinatoric problems along side the high activity in this event class make a classical cut-and-count analysis difficult. This makes $t\bar{t}H$ a great candidate to explore the capsule networks capabilities.

Based on our experiences with the $Z' \to t\bar{t}$ classification, our first instinct was to employ the pooling architecture that was so successful previously, for the $t\bar{t}H$ case as well. However, the 'pooling' setup is not appropriate here, since the reduced resolution caused by the pooling is problematic for $t\bar{t}H$ tagging. For this reason we use the 'conv' architecture, which uses the fully resolved images, and used it as a basis for our $t\bar{t}H$ network, shown in Fig. 47.

The first change compared to the original 'conv' architecture in Fig. 34 is that we doubled the amount of filters in each intermediate convolutional layer. This has little impact on the total amount of trainable network parameters, since the bulk of the networks parameters are in the capsule layers, rather than the convolutions, but gives us a better handle for extracting more detailed image features.

Further we were looking for ways of giving the network additional physical information to improve classification power. To this end, we use the information we assumed to have available during the pre-selection cuts. Specifically, we make use of the fact that we already demand four tagged b-jets, six total jets and one tagged lepton to be present in the event and can therefore pass this information along to the network without making additional assumptions. Our way of storing this information is by creating three additional 180×180 pixel images, where, at the position of a b-jet center, we place a pixel entry whose value is equal to the jets transversal energy. For the two remaining images we do the same except with light-jets and leptons respectively. We then pass these additional feature maps alongside the main calorimeter image.

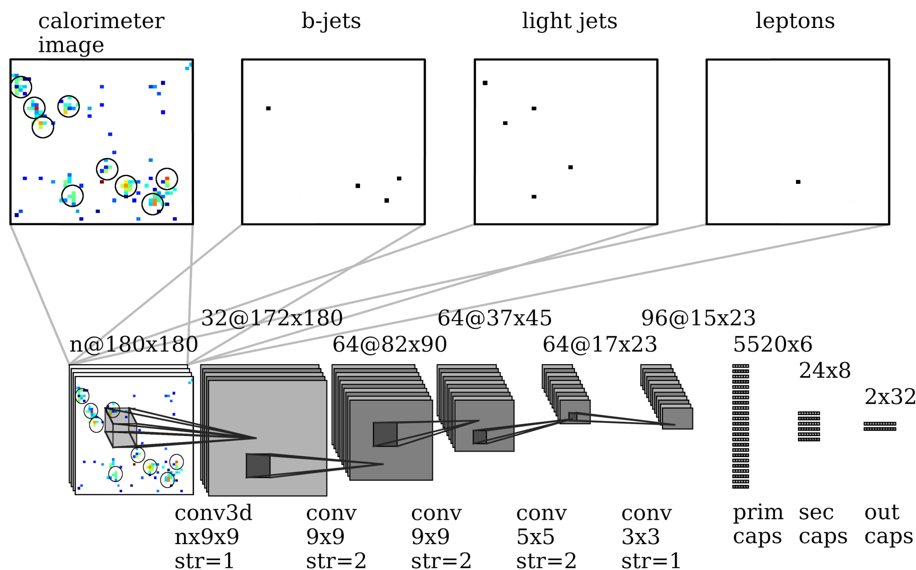

Figure 47: Capsule network for $t\bar{t}H$ tagging, with 3D-convolution to process additional information about jet flavors. As seen in Ref. [25].

To allow the network to handle the additional information we use a 3D-convolution to on in the input layers. A 3D-convolution is based on the same principle as a the 2D-convolution described in section 3.1.3, however input, output and convolution-kernel are all rank-3 tensors instead of matrices. As our 3D input we use the stack of input layers. This has several advantages compared to, for example, processing the additional input layers by passing each through a separate convolutional structure in parallel and then recombining the convolution outputs before the first capsule layer (similar to the concatenation layer in Fig. 38). The parallel setup would lose some information about the spacial relationship between the different input layers since it looks at them individually. The 3D-convolution, however, lets the network learn relations between different input features that occupy the same position in the η-ϕ-plane. For example, the network can use this to associate a b-tag in the b-jets input layer with an actual jet in the calorimeter image. Since we want to have the network take all input layers into account simultaneously we choose the depth of the 3D-kernel to be equal to the amount of input layers, as depicted in Fig. 47, so for our four layer example we would use a $9 \times 9 \times 4$-kernel. This has the added benefit of producing a 2D rather then 3D output, which allows us to directly use the output of the 3D convolution as an input for the following 2D-convolutional layers.

With this setup our capsule network is now capable of classifying $t\bar{t}H$ events, as we can see in Fig. 48. Here, we compare three levels of information. The first, blue curve shows the results using just the calorimeter image. For this we achieve an AUC of 0.715, which is comparable to the machine learning $t\bar{t}H$ analysis in [61]. Next, we give the network additional

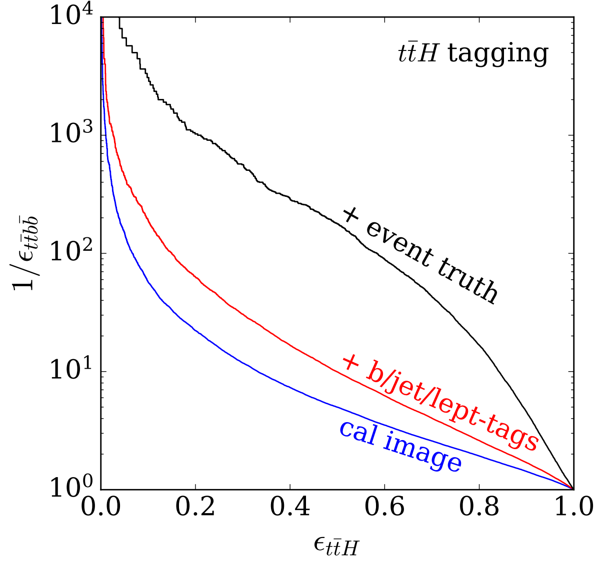

Figure 48: ROC curves comparing the performance of the $t\bar{t}H$ capsule network with different information levels. Taken from Ref. [25].

feature maps marking the positions of b-jets, light jets and leptons. This result is shown in the red curve in Fig. 48 and as we can see, the added information greatly improves the performance and giving us an AUC of 0.792. From this, we can also see that the capsule network is indeed capable of putting the additional input layers to good use. Finally, for every signal versus background classification there will be an upper limit on how accurate an analysis can be, simply because some signal and background events will look identical and therefore be indistinguishable. In order to estimate this limit we use the black curve in Fig. 48. This time we gave the network even more feature maps, with all the information needed to solve the combinatorics problem. Specifically the layers marked the positions of: light-jets, leptons, b-jets, $\bar{b}$-jets, b-jet from top decays and $\bar{b}$-jets from anti-top decays. With this, the network can easily identify the two b-jets that potentially originate from a Higgs decay and use their invariant mass for classification, similar to what was shown in Fig. 45. This, as expected, adds a huge boost to the networks performance and results in an AUC of 0.927. From this result, we can deduce two things, for one there seems to be a quite big overlap between signal and background, which even with solved combinatorics could not be separated. This illustrates how similar the $t\bar{t}H$ and $t\bar{t}b\bar{b}$ events are. Further, we can see that the capsule network is capable of differentiating signal and background very well, when given sufficient information. The lack of this combinatoric information is the main bottleneck of this analysis.

Finally, we have seen that capsule networks are indeed able to tag even complex, high-activity events, such as $t\bar{t}H$ against very similar backgrounds and that, by using 3D-convolutions, we have a great handle for giving the network additional information.

9 Conclusion and Outlook

Over the course of this thesis we have employed capsule network setups to classify whole LHC events using several different event types.

Initially we have shown that capsule networks are a powerful tool for particle physics. Through their vector based approach to neural networking, capsules are not only capable of tagging individual jet images, but are also able to overcome the difficulties that arise when classifying whole LHC events.

By both successfully differentiating $t\bar{t}$ events with obscured event level features, from QCD di-jet events, as well as separating $Z' \to t\bar{t}$ from QCD mediated $t\bar{t}$ events, we demonstrated that it is possible for capsule networks to learn sub-jet features as well as event level kinematics. Further, we proved the network is capable of utilizing both information levels simultaneously by separating a mixed background sample. Additionally, we were able to make use of the unique structure of the capsules to understand how the network learns and what it uses for classification.

Finally, we demonstrated that the capsule approach remains viable, even for complex, high-activity events such as $t\bar{t}H$, where we were also able to boost our network performance trough additional input images.

Although we have seen the capsule setup perform very well in combination with convolutions, the idea of capsules is not limited to them. Future capsule network applications could include a combination of point-cloud, edge convolution structure [62] and capsules, which appears highly promising. Additionally, one could apply the same ideas employ in Bayesian neural networks [63] to capsules networks, enabling per-event uncertainty predictions while using capsules.

References

[1] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. 1997.

[2] Josh Cogan, Michael Kagan, Emanuel Strauss, and Ariel Schwarztman. Jet-images: computer vision inspired techniques for jet tagging. *Journal of High Energy Physics*, 2015:118, Feb 2015.

[3] Luke de Oliveira, Michael Kagan, Lester Mackey, Benjamin Nachman, and Ariel Schwartzman. Jet-images — deep learning edition. *Journal of High Energy Physics*, 2016(7):69, Jul 2016.

[4] James Barnard, Edmund Noel Dawe, Matthew J. Dolan, and Nina Rajcic. Parton shower uncertainties in jet substructure analyses with deep neural networks. *prd*, 95(1):014018, Jan 2017.

[5] V. Mohanraj, S. Sibi Chakkaravarthy, and V. Vaidehi. Ensemble of convolutional neural networks for face recognition. In Jugal Kalita, Valentina Emilia Balas, Samarjeet Borah, and Ratika Pradhan, editors, *Recent Developments in Machine Learning and Data Analytics*, pages 467–477, Singapore, 2019. Springer Singapore.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. *arXiv e-prints*, page arXiv:1604.07316, Apr 2016.

[7] Jason Gallicchio and Matthew D. Schwartz. Quark and gluon jet substructure. *Journal of High Energy Physics*, 2013:90, Apr 2013.

[8] Patrick T. Komiske, Eric M. Metodiev, and Matthew D. Schwartz. Deep learning in color: towards automated quark/gluon jet discrimination. *Journal of High Energy Physics*, 2017(1):110, Jan 2017.

[9] Taoli Cheng. Recursive Neural Networks in Quark/Gluon Tagging. *arXiv e-prints*, page arXiv:1711.02633, Nov 2017.

[10] Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler. Energy flow networks: deep sets for particle jets. *Journal of High Energy Physics*, 2019(1):121, Jan 2019.

[11] Samuel Bright-Thonney and Benjamin Nachman. Investigating the topology dependence of quark and gluon jets. *Journal of High Energy Physics*, 2019(3):98, Mar 2019.

[12] Gregor Kasieczka, Nicholas Kiefer, Tilman Plehn, and Jennifer Thompson. Quark-gluon tagging: Machine learning vs detector. *SciPost Physics*, 6(6):069, Jun 2019.

[13] Sung Hak Lim and Mihoko M. Nojiri. Spectral analysis of jet substructure with neural networks: boosted Higgs case. *Journal of High Energy Physics*, 2018(10):181, Oct 2018.

[14] Amit Chakraborty, Sung Hak Lim, and Mihoko M. Nojiri. Interpretable deep learning for two-prong jet classification with jet spectra. *Journal of High Energy Physics*, 2019(7):135, Jul 2019.

[15] Leandro G. Almeida, Mihailo Backović, Mathieu Cliche, Seung J. Lee, and Maxim Perelstein. Playing tag with ANN: boosted top identification with pattern recognition. *Journal of High Energy Physics*, 2015:86, Jul 2015.

[16] Jannicke Pearkes, Wojciech Fedorko, Alison Lister, and Colin Gay. Jet Constituents for Deep Neural Network Based Top Quark Tagging. *arXiv e-prints*, page arXiv:1704.02124, Apr 2017.

[17] Shannon Egan, Wojciech Fedorko, Alison Lister, Jannicke Pearkes, and Colin Gay. Long Short-Term Memory (LSTM) networks with jet constituents for boosted top tagging at the LHC. *arXiv e-prints*, page arXiv:1711.09059, Nov 2017.

[18] Suyong Choi, Seung J. Lee, and Maxim Perelstein. Infrared safety of a neural-net top tagging algorithm. *Journal of High Energy Physics*, 2019(2):132, Feb 2019.

[19] Liam Moore, Karl Nordström, Sreedevi Varma, and Malcolm Fairbairn. Reports of My Demise Are Greatly Exaggerated: N-subjettiness Taggers Take On Jet Images. *arXiv e-prints*, page arXiv:1807.04769, Jul 2018.

[20] Gregor Kasieczka, Tilman Plehn, Michael Russell, and Torben Schell. Deep-learning top taggers or the end of QCD? *Journal of High Energy Physics*, 2017(5):6, May 2017.

[21] Anja Butter, Gregor Kasieczka, Tilman Plehn, and Michael Russell. Deep-learned Top Tagging with a Lorentz Layer. *SciPost Physics*, 5(3):028, Sep 2018.

[22] Sebastian Macaluso and David Shih. Pulling out all the tops with computer vision and deep learning. *Journal of High Energy Physics*, 2018(10):121, Oct 2018.

[23] Reza Katebi, Yadi Zhou, Ryan Chornock, and Razvan Bunescu. Galaxy morphology prediction using Capsule Networks. *mnras*, 486(2):1539–1547, Jun 2019.

[24] V. Lukic, M. Brüggen, B. Mingo, J. H. Croston, G. Kasieczka, and P. N. Best. Morphological classification of radio galaxies: capsule networks versus convolutional neural networks. *mnras*, 487(2):1729–1744, Aug 2019.

[25] Sascha Diefenbacher, Hermann Frost, Gregor Kasieczka, Tilman Plehn, and Jennifer M. Thompson. CapsNets Continuing the Convolutional Quest. *arXiv e-prints*, page arXiv:1906.11265, Jun 2019.

[26] A Butter, G Kasieczka, T Plehn and M Russell. Top Tagging Reference Dataset, 2017-NNNN. https://goo.gl/XGYju3.

[27] CMS Collaboration. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716(1):30–61, Sep 2012.

[28] ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1–29, Sep 2012.

[29] J. M. Campbell, J. W. Huston, and W. J. Stirling. Hard interactions of quarks and gluons: a primer for LHC physics. *Reports on Progress in Physics*, 70(1):89–193, Jan 2007.

[30] CMS Collaboration. Particle-flow reconstruction and global event description with the CMS detector. *Journal of Instrumentation*, 12(10):P10003, Oct 2017.

[31] Simone Marzani, Gregory Soyez, and Michael Spannowsky. Looking inside jets: an introduction to jet substructure and boosted-object phenomenology. *arXiv e-prints*, page arXiv:1901.10342, Jan 2019.

[32] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. The anti-k_t jet clustering algorithm. *Journal of High Energy Physics*, 2008(4):063, Apr 2008.

[33] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. *arXiv e-prints*, page arXiv:1412.6806, Dec 2014.

[34] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexand re G. R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. *physrep*, 810:1–124, May 2019.

[35] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964.

[37] Y.E. Nesterov. A method for solving the convex programming problem with convergence rate o(1/k2). *Soviet Mathematics Doklady*, 27:372–376, 1983.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec 2014.

[39] Max Pechyonkin. Understanding Hintons Capsule Networks. Part 1. Intuition., 2017-NNNN. https://pechyonkin.me/capsules-1/.

[40] Xifeng Guo. A Keras implementation of CapsNet in NIPS2017 paper "Dynamic Routing Between Capsules", 2017-NNNN. https://github.com/XifengGuo/CapsNet-Keras.

[41] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic Routing Between Capsules. *arXiv e-prints*, page arXiv:1710.09829, Oct 2017.

[42] T. Gleisberg, S. Höche, F. Krauss, M. Schönherr, S. Schumann, F. Siegert, and J. Winter. Event generation with SHERPA 1.1. *Journal of High Energy Physics*, 2009(2):007, Feb 2009.

[43] Tanju Gleisberg and Stefan Höche. Comix, a new matrix element generator. *Journal of High Energy Physics*, 2008(12):039, Dec 2008.

[44] Richard D. Ball, Valerio Bertone, Stefano Carrazza, Christopher S. Deans, Luigi Del Debbio, Stefano Forte, Alberto Guffanti, Nathan P. Hartland, José I. Latorre, Juan Rojo, and Maria Ubiali. Parton distributions for the LHC run II. *Journal of High Energy Physics*, 2015:40, Apr 2015.

[45] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi. DELPHES 3: a modular framework for fast simulation of a generic collider experiment. *Journal of High Energy Physics*, 2014:57, Feb 2014.

[46] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. FastJet user manual. (for version 3.0.2). *European Physical Journal C*, 72:1896, Mar 2012.

[47] Matteo Cacciari and Gavin P. Salam. Dispelling the N^3 myth for the k_t jet-finder. *Physics Letters B*, 641(1):57–61, Sep 2006.

[48] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth.*, A389:81–86, 1997.

[49] The HDF Group. Hierarchical Data Format, version 5, 1997-NNNN. http://www.hdfgroup.org/HDF5/.

[50] Gregor Kasieczka, Tilman Plehn, Anja Butter, Kyle Cranmer, Dipsikha Debnath, Barry M. Dillon, Malcolm Fairbairn, Darius A. Faroughy, Wojtek Fedorko, Christophe Gay, Loukas Gouskos, Jernej Fesel Kamenik, Patrick Komiske, Simon Leiss, Alison Lister, Sebastian Macaluso, Eric Metodiev, Liam Moore, Benjamin Nachman, Karl Nordström, Jannicke Pearkes, Huilin Qu, Yannik Rath, Marcel Rieger, David Shih, Jennifer Thompson, and Sreedevi Varma. The Machine Learning landscape of top taggers. *SciPost Physics*, 7(1):014, Jul 2019.

[51] François Chollet et al. Keras. `https://keras.io`, 2015.

[52] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner,

Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[53] Martin Bauer, Sascha Diefenbacher, Tilman Plehn, Michael Russell, and Daniel A. Camargo. Dark matter in anomaly-free gauge extensions. *SciPost Physics*, 5(4):036, Oct 2018.

[54] Adam Alloul, Neil D. Christensen, Céline Degrande, Claude Duhr, and Benjamin Fuks. FEYNRULES 2.0 - A complete toolbox for tree-level phenomenology. *Computer Physics Communications*, 185(8):2250–2300, Aug 2014.

[55] Neil D. Christensen and Claude Duhr. FeynRules - Feynman rules made easy. *Computer Physics Communications*, 180(9):1614–1641, Sep 2009.

[56] Céline Degrande, Claude Duhr, Benjamin Fuks, David Grellscheid, Olivier Mattelaer, and Thomas Reiter. UFO - The Universal FEYNRULES Output. *Computer Physics Communications*, 183(6):1201–1214, Jun 2012.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[58] L. E. Blumenson. A derivation of n-dimensional spherical coordinates. *The American Mathematical Monthly*, 67(1):63–66, 1960.

[59] Tilman Plehn, Michael Spannowsky, Michihisa Takeuchi, and Dirk Zerwas. Stop reconstruction with tagged tops. *Journal of High Energy Physics*, 2010:78, Oct 2010.

[60] Tilman Plehn, Peter Schichtel, and Daniel Wiegand. Where boosted significances come from. *prd*, 89(5):054002, Mar 2014.

[61] M. Erdmann, E. Geiser, Y. Rath, and M. Rieger. Lorentz Boost Networks: autonomous physics-inspired feature engineering. *Journal of Instrumentation*, 14(6):P06006, Jun 2019.

[62] Huilin Qu and Loukas Gouskos. ParticleNet: Jet Tagging via Particle Clouds. *arXiv e-prints*, page arXiv:1902.08570, Feb 2019.

[63] Sven Bollweg, Manuel Haussmann, Gregor Kasieczka, Michel Luchmann, Tilman Plehn, and Jennifer Thompson. Deep-Learning Jets with Uncertainties and More. *arXiv e-prints*, page arXiv:1904.10004, Apr 2019.

Acknowledgments

First of all, I would like to thank Prof. Dr. Tilman Plehn, both for allowing me to work in his research group on such an interesting topic and for providing 'motivation' in necessary cases.

Further I would like extend my sincerest gratitude to Dr. Jennifer Thompson, for frequent council, be it regarding machine learning, event generation or physics in general, as well as for taking their time to proof read this thesis.

Similarly I would like for Michel Luchman and Anja Butter to know, their great advice on many different physics topics was highly appreciated.

In the same vein I would like to thank both Prof. Dr. Gregor Kasieczka for great input on machine learning problems, as well as thank both him and Hermann Frost for figuring out the interpretation of the capsule entries.

And finally I want to thank the entire Heidelberg LHC Physics and New Particles research group, not only for their frequent help with questions, but also for making my time there enjoyable.

Lastly I acknowledge support by the state of Baden-Württemberg through bwHPC.

Erklärung

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

Heidelberg, den 12.09.2019

Sascha Daniel Diefenbacher