1 Convolutional Autoencoder Trigger Concept

In the second part of this thesis a concept of trigger primitive generation for the HGCAL detector is presented. The concept is based on the theory of neural networks, a subdomain of artificial intelligence, exploring mathematical structures loosely inspired by the brain. The first section of this chapter introduces the general concept and working principles of neural networks (NN), followed by the presentation of two specific types of NNs, autoencoders and convolutional networks, together with the popular MNIST dataset used as surrogate data. These three ingredients are of central importance to the concept developed in the subsequent part, after which studies on conceptual aspects with respect to the HGCAL environment are presented. This chapter is completed with a discussion in which all results are reviewed in view of their application to the concept.

1.1 Neural network

A neural network is a mathematical structure embedding a functional form. In a process called neural network training, certain network parameters, i.e. coefficients of the functional form, are gradually adjusted such that the network function approximates a desired function, which is usually not known analytically. In principle, any function can be fitted arbitrarily close [**?**] depending on the network capacity (amount of parameters), its architecture, the training time and amount of available (and relevant) data. Thus, the goal of training is to find a function underlying a problem such that the network is able to predict the correct solution for new data that belongs to the same problem domain, i.e. the network is able to generalize beyond the training data. However, neural networks are not always guaranteed to reach generalization power after training.

Neural networks are composed of neurons (nodes) and weights (edges) that are arranged in consecutive layers (see Fig. 1) such that each layer's neurons receive their inputs from the previous layer's neurons and send their outputs to the successive layer's neurons. Data is fed to the network via the first (input) layer, undergoes transformations in the intermediate (hidden) layers until it arrives at the final (output) layer that can be read out and interpreted. The amount of hidden layers separates shallow neural networks from deep networks. Deep networks can easily span tens or hundreds [**?**] of hidden layers.

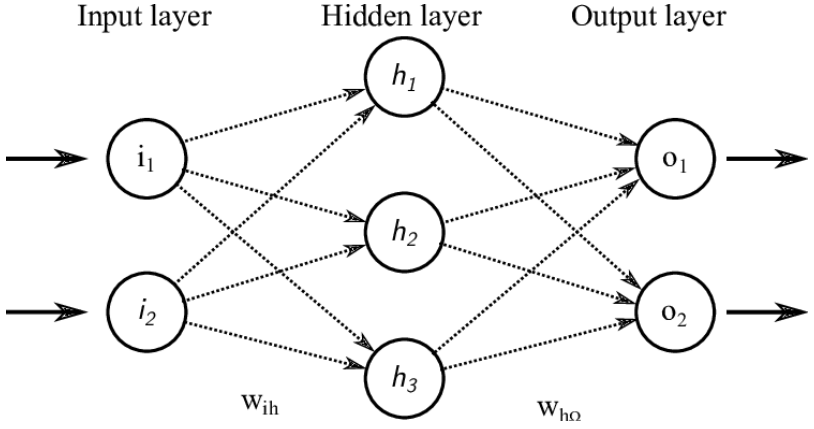


Figure 1: Simple neural network architecture. Figure taken from Ref. [**?**].

Data propagating through a neural network is successively transformed by the neurons of each layer. Each neuron transforms its inputs into a single output value, called *activation*. In a fully connected neural network architecture (Fig. 1) the activation of a single neuron is defined by

$$a(\vec{x}) = \sigma\left(\sum_i w_{ih} x_i + b\right), \tag{1}$$

where $\vec{x}$ represents the inputs (i.e. previous layer's outputs), w_{ih} the connecting weights and b the bias term. The weights and biases are the free parameters of the network, usually initialized to random values before adjustment due to network training. σ is a non-linear function (tanh, sigmoid, ReLU, softmax, etc.) that allows the model to capture non-linear relationships in the input data. Geometrically, a neuron activation transforms the input in three ways:

1. Transform the input linearly: $\sum_i w_{ih} x_i$.
2. Translate to a different position by b.
3. Warp by σ. This effectively maps the input into a non-linear manifold.

Successive non-linear layers allow complex transformations [**?**]. These networks are referred to as deep neural networks (DNNs). Non-linearity is a decisive criterion for depth since a stack of linear hidden layers, on the other hand, can be collapsed to a single layer containing linear combinations of the former. Therefore, DNNs are much more powerful than shallow (linear) nets.

Since successive hidden layers obtain their inputs from previous layers, $\vec{x}$ can be replaced by $\vec{a}^{l-1}$, where the subscript l refers to the depth index of a layer. Ordering neurons in layers further allows to elevate Eq. 1 to a vector-matrix equation:

$$\vec{a}^{l}(\vec{a}^{l-1}) = \sigma\big(\mathbf{W}^{l} \cdot \vec{a}^{l-1} + \vec{b}^{l}\big), \tag{2}$$

with the weight matrix $\mathbf{W}^{l}$ between two consecutive layers and $\vec{b}^{l}$ being a layer's biases. In this form the (forward) propagation of data through the network can be computed rather easily by basic linear algebra subprograms that are highly optimized for matrix multiplications. A graphical processing unit (GPU) can further accelerate the computation by introducing many computational cores that allow for massive parallelization.

As network training is a curve fitting process involving many free parameters, care must be taken to not overfit (or underfit) the model on the training data. Overfitting happens when the parameters are tuned too much. The desired function is for example linear while the network learned a quadratic fit due to the presence of noise in the training data, i.e. the network learns a fit to random fluctuations caused by noise which do not capture the underlying problem. Underfitting is the reverse process of a linear fit to a desired quadratic function resulting from too few network parameters or insufficient training time. To prevent overfitting or underfitting, the training process should be monitored and halted at the right time. Additionally, so-called regularization techniques can be used to prevent overfitting (and also accelerate the training, i.e. the convergence of the fit). More details on networks training can be found in Appendix **??**.

The training process is guided by a so-called loss (or objective) function $\mathscr{L}$. Since neural networks can be trained for different purposes (classification, regression, reconstruction, data generation, etc.) a multitude of loss functions can be found in the literature. For a given input, the loss function takes the activations of the network's output layer and a corresponding desired output (ground truth) as input and generates a single real number to be minimized. For reconstruction, for example, the mean-squared error function

$$\mathscr{L}_{MSE}(\vec{x}, \vec{y}) = \sum_{n} (x_i - y_i)^2 \tag{3}$$

is often used. If the desired output (truth) has been generated in advance by experts or external sources one speaks of supervised learning. One variant of supervised learning is self-supervised learning in which the ground truth and input data are identical.

In the following sections special network architectures are discussed, that aide in the subsequent presentation of a neural network based trigger primitive generator for the HGCAL detector.

1.1.1 Autoencoder

An autoencoder is a type of neural network which is often used for dimensionality reduction. By introducing a bottleneck in the architecture of the network, a lower dimensional representation (encoding) of input data is achieved, that is subsequently transformed back into the original input space (decoding). The decoded reconstruction of the input is compared to the original input via a reconstruction error, often the mean squared error $MSE = \frac{1}{n}\sum_i^n (x_i - \hat{x}_i)^2$, for n features x_i of a data point and its corresponding reconstruction $\hat{x}_i$. By minimizing the reconstruction error, the network seeks for the best possible representation of the input in its bottleneck (also referred to as latent space).

The simplest type of autoencoder is the *undercomplete* autoencoder, depicted in Fig. 2. The term undercomplete refers to the latent space, which is smaller in dimensions than the input space. Other types of autoencoders, on the other hand, can be *over*complete. They initially contain a latent space larger than the input space that is subsequently reduced by training under regularization constraints that penalize the activation of latent neurons and thus creating the bottleneck. These types include the sparse, contractive, and denoising autoencoders. Variational autoencoders, on the other hand, are a network architecture for data generation. All of these can be further complemented with a deep stack and/or convolutional layers giving rise to two further categories: Deep and convolutional autoencoders. Deep undercomplete convolutional autoencoders are considered in this thesis.

When using deep autoencoders with nonlinear activation functions, the network is able to learn a more powerful generalization of the (linear) dimensionality reduction technique Principle Component Analysis (cf. Appendix **??**), since nonlinearities allow for warping as discussed in the beginning of Sec. 1.1 which can account for complex relationships in the input data by allowing transformations to more complex manifolds. Furthermore, [**?**] reported that depth can exponentially reduce computational cost and the amount of training data needed to learn certain functions. In order to achieve generalization in deep autoencoders overfitting must be prevented: Too much capacity (layer depth and

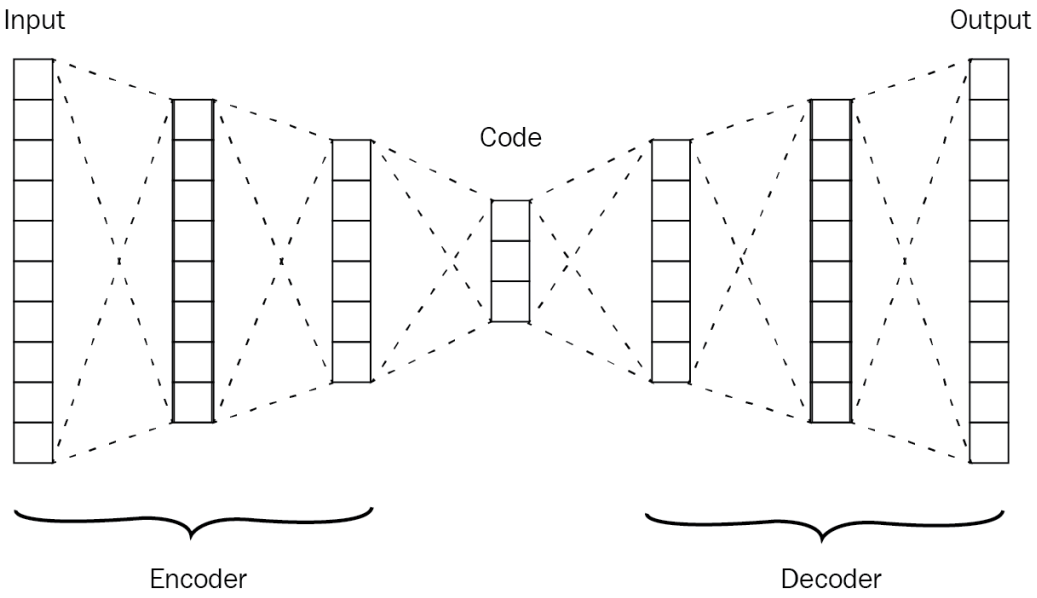


Figure 2: Schematic of a five-hidden-layer deep undercomplete autoencoder network. Figure taken from Ref. [**?**].

width) could lead to learning an identity function by mapping each sample of the training dataset to an index in latent space (a single hidden neuron would be sufficient) from which the input is perfectly reconstructed – a reconstruction error of zero. In this case, the network would extremely overfit on the training data and yield no useful generalization [**?**]. Therefore, a common practice for training deep autoencoders is *greedy pretraining*, in which the layers of the encoder and decoder parts are trained successively in isolation on each others latent representations before combining them to the deep model.

1.1.2 Convolutional Neural Network

Convolutional neural networks (CNNs) are neural networks containing (a stack of) convolutional layers, which are particularly well-suited when processing image-like data. Real-world images often exhibit an inherent bias – a spatial relationship between the features in a scene. A digit, for example, can be interpreted as a composition of simple features: straight lines, round edges and circles arranged in a certain way. CNNs use (small) filters to learn these features by processing partial areas of the input data at a time. By sliding the filter (also called kernel) across the image distinct features can be learned, and later (at inference time) detected equally well in all parts of the image. This technique makes CNNs invariant to the position where a feature appears in the image.

Figure 3 shows the working principle of the convolutional layer. The pixel values of an image patch are weighted by a filter and afterwards added together to form the output activation of a resulting image (called feature map). Subsequently, the window over the input image is advanced and another weighted sum is calculated to build the second activation of the feature map. This process is repeated until all patches of the image have a corresponding activation on the feature map. The process of "sweeping" a kernel K along an image I is closely related to the convolution operation $K * I$. Since K is not flipped before sweeping this operation is, more precisely, the cross-correlation operation.

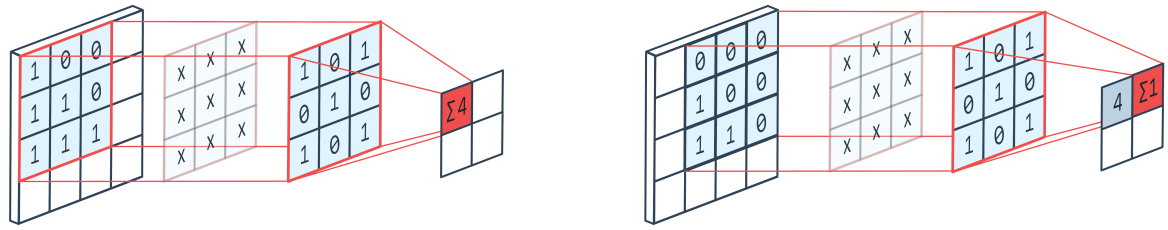


Figure 3: Working principle of convolutional layers. One pixel value of the resulting feature map is a weighted sum of an image patch with the kernel values. Figure taken from Ref. [**?**].

The size of the feature map is determined by three parameters: Filter size, step size (stride), and the amount of additional padding. In some cases where it is desired to preserve the original input image dimensions in the resulting feature map, a stride size of one in x- and one in y-direction can be chosen, while zero padding the image according to the kernel size, which is referred to as *same* padding. *Valid* padding on the other hand does not pad the image and also only allows windows that are fully inside the input image. The output (feature map) dimensions n_{out} of a convolutional layer can thus be calculated from the input dimensions n_{in}, the amount of padding to one side P, the step size (strides) S and the kernel size K as

$$n_{out} = \left\lfloor \frac{n_{in} + 2P - K}{S} \right\rfloor + 1, \tag{4}$$

In CNNs several convolutional layers are used in succession, thus subsequent feature maps take preceding maps as inputs to look for local features. To illustrate the effect, consider a dataset of handwritten digits. While the first convolutional layer learns (low-level) features like edges and shapes, showing their presence as high activations in the corresponding feature map, the second convolutional layer has the ability to combine the presence of these features to find more general (higher-level) features, i.e. to combine for example the activations of a straight line and a circle in distinct parts of the input image to match activations corresponding to the digit *9*. Usually, several filters are learned in parallel for one convolutional layer.

The trainable parameters of convolutional layers are the weights of the kernel together with a bias. Since kernels are usually small and only the weights of the kernel are learned, the number of network parameters is much smaller compared to fully connected networks, which makes CNNs both fast (if parallelized) and memory efficient.

In the last decade, CNNs have been extensively researched, so that they became a de facto standard when dealing with image-like data in neural networks.

1.1.3 MNIST dataset

The MNIST dataset is a collection of 70 000 scanned images of handwritten digits, together with their correct classification. The name of the dataset comes from the fact

that it is composed of a **m**odified subset[1] of two datasets collected by the United States' **N**ational **I**nstitute of **S**tandards and **T**echnology. Some of the images contained in the dataset can be seen in Fig. 4. The 70 000 images are split in two parts of 60 000 for training and 10 000 for testing. Training and test set are composed of writing samples from two different groups of people in order to allow testing the network independently sampled data is not used during training [**?**].

Figure 4: Subset of the MNIST handwritten digit dataset. Figure taken from Ref. [**?**].

The data points of the MNIST dataset are 28×28 greyscale images, in which each pixel has an integer intensity between 0 (black) and 255 (white). The correct classification of training and test data comes in two companion datasets, which are composed of integers between zero and nine that represent the class of the digit shown on the image. In this thesis, the MNIST dataset was used as a surrogate for calorimetric shower data because of the following reasons:

- Granular calorimeter data is in principle image-like, since shower shapes exhibit spatial information.
- Like handwritten digits, shower shapes have semantic meaning, i.e. their energy content, type of shower-originating particle, and its incident location.
- The sensor cell reading is, like greyscale images, single-channel data.
- Interpretations of classification and reconstruction of handwritten digits is closer to the domain of human experience than doing the same on shower data. Thereby, experimentation results can be generated faster and are easier to interpret.
- Also, MNIST is a complete and well-studied dataset that can be used directly.

[1] Size-normalized and centered in a fixed-size image frame [**?**].

1.2 Concept studies

While the previous sections introduced the basics of neural networks, special architectures and a popular dataset, the following paragraphs discuss how these techniques can be used inside HGCAL's trigger primitive generator that was introduced in Section **??**. In particular, it will be illustrated how neural networks can be exploited for the purpose of compressing data and extracting relevant information in order to supply the Level-1 trigger system with the necessary information to make a trigger decision.

When bunches of particles cross at the intersection point of the CMS experiment, a subset of these particles are subject to hard collisions, producing a range of other particles in the process, of which a few interact in the end-cap calorimeters. Particles produce electromagnetic or hadronic showers, described in Section **??**, which traverse the detector. The shapes of these showers often differ for different types of incident particles and energies. In order to generate a trigger decision, i.e. to determine if a bunch crossing contained collisions that are relevant for physics processes the CMS physics program is looking for, the trigger system must distill three types of information from the sensor data for each shower that has been recorded:

1. Type of incident particle that induced a shower, e.g. γ, $e^{\pm}$, π, τ, etc.
2. Energy of the incident particle (inferred from shower energy).
3. Hit position and direction of incident particle.

Since the Level-1 trigger decision must be generated for every BX in a short amount of time the data rate that the L1T algorithms can process is limited. Therefore, sensor data is compressed before processing, with the goal to conserve as much information as possible. Current implementations of compression algorithms in the first of three stages of compression in the HGCAL TPG, the **e**lectronic **con**centrator chip for **t**rigger data (ECON-T), shows varying performance depending on the type and incident position of particles. This situation is summarized in Fig. 5, which shows the triggger rate as a function of the offline threshold[2] for different incident particles. A low trigger rate corresponds to better classification accuracy on the given data. An increase in rate means that an algorithm needs more data in order to reach the same classification accuracy. Thus, there is no single algorithm which performs well in all cases. Furthermore, the large amount of expected pile-up in the high luminosity environment of Run 4 worsens the situation significantly due to overlapping showers and an increase in ambiguities in shower data.

These facts lead to the question of whether there exists a better algorithm which combines the advantages of the above algorithms, yielding good classification performance on all types of incident particles. As discussed in Section 1.1.1 autoencoders are a higher-level generalization of conventional compression algorithms that allow more powerful representations by exploiting more complex relationships in the input data. The idea, introduced in this chapter is to develop a neural network based system spanning the three levels of compression (ECON-T, Stage 1, Stage 2) of the HGCAL TPG to mitigate the problem of particle type dependent trigger rates. In this thesis, several neural network models have been developed in order to investigate the impact of architectural boundary conditions

[2]The offline threshold is defined as the threshold that achieves 95% trigger efficiency.

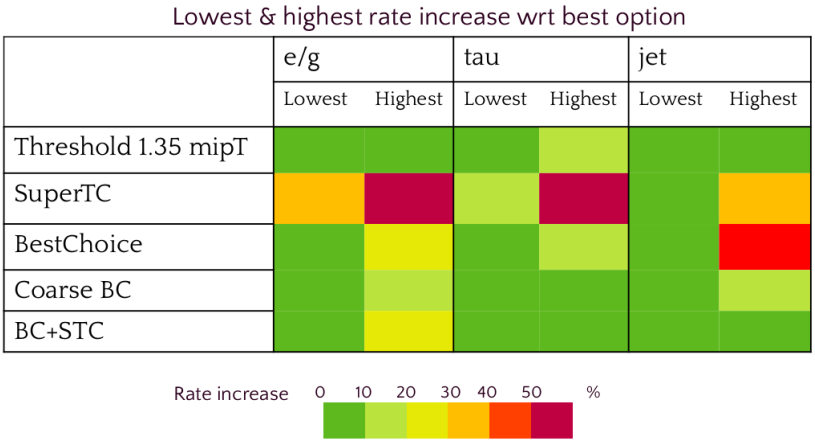


Figure 5: Increase of trigger rate rate as a function of offline threshold for the three different compression algorithms of the ECON-T chip: Threshold, Super Trigger Cell, BestChoice. These algorithms have been presented in Sec. **??**. Two further variants Coarse BestChoice first applies sorting and then selects the N largest values, BC+STC uses BestChoice in the ECAL and Super Trigger Cells in the HCAL, based on the observation that BestChoice works best for EM showers (small objects) and STC best for jets (larger objects). Table taken from Ref. [**?**].

imposed by the HGCAL TPG to support the development of a proof-of-concept. These boundary conditions are:

- **Elementary encoder unit**: The recently integrated generic encoder block in the ECON-T allows an implementation of a concise hardware-based neural network. As discussed in the previous sections, autoencoders are particularly well suited for finding meaningful encodings that are a generalization of other encoding techniques.

- **Split encoding**: Shower data is distributed among several ROCs – each ROC only sees an excerpt of the information it is meant to preserve. Thus, the question arises as to how effectively encoders can compress parts of a scene while still allowing for the desired information to be recovered by combining the corresponding encodings, i.e. how much does a global encoding differ from the combination of local encodings?

- **Invariance in ϕ**: Physical processes inside the detector are identical for same η, in concentric rings around the beam axis. Thus, showers induced by same particles with same energies are expected to follow the same statistics. By this virtue, encoding units in these rings can be expected to behave very similarly.

- **Partial encoding**: Shower-inducing particles can hit the detector at different positions. Thus, the occupancy of sensors can look vastly different from the viewpoint of the encoder units. In order to account for this, the units must be trained on images that exhibit this randomness in location.

- **Trigger primitive information**: As discussed in Sec. **??** the longitudinal and lateral extent, as well as the internal structure of a shower contain information about the three relevant types of information: energy, particle type, and incident position. This is the primary information that is desired to be preserved.

- **Information bottleneck**: Because of the timing constraint for L1T decisions the sensor data needs to be reduced by a factor of $5 \times 30 \times 8 = 1200$ in the three[3] compression stages of the HGCAL trigger primitive generator (cf. Sec. **??**). Thus, to make a trade-off decisions, a relationship between compression ratio and information capacity would be useful.

Based on these constraints a neural network architecture was developed, schematically depicted in Fig. 6. It shows three stages of feed-forward encoder networks. The corresponding decoders will be implemented in software for training and monitoring, thereby allowing for sparser hardware implementations of just the encoder part. The network as a whole is trained such that relevant information is maximally preserved. In this thesis the key conceptual ideas are explored by using a smaller scale architecture on the toy problem of recognizing handwritten digits from the MNIST dataset. In the following, these studies will be presented togehter with their results.

1.2.1 Elementary encoder unit

As shown in Fig. 6, the concept is built on multiple elementary encoder units successively propagating encoded latent representations through a cascade of encoding stages, starting from calorimetric (trigger cell) data and ending with trigger primitives. The following studies focus on the behavior of encoders in a single stage (ECON-T) by considering all of the aforementioned boundary conditions. The cascading behavior of multiple consecutive stages is subject of future studies. The subject of this section is the basic building block of the concept, the elementary encoding unit, and its reconstruction performance under ideal conditions.

Calorimetric data is inherently image-like – sensor cell readings can be interpreted as pixels (of energy intensity) forming an image of a shower. As discussed in Sec. **??**, the length, width, and overall shape of showers carries macroscopic information that allows to infer information about its inducing particle. Thus, the encoder unit should be capable of extracting structural information from the shower image. Sec. 1.1.2 highlighted that convolutional neural networks are the usual choice when it comes to such data due to their ability to extract a hierarchy of features by correlating features in a layer with themselves (inductive bias).

Additionally, each encoding stage has the task to compress the calorimetric data by a certain ratio without losing too much information. As discussed in Sec. 1.1.1, the goal of PCA is to preserve dimensions (features) in the data that show the most variance, thus carry the most information and that autoencoders are a more powerful generalization of PCA. Thus, an autoencoder architecture is the preferred choice for the encoder networks. Furthermore, since the compression stages have a fixed amount of incoming and outgoing data links, the implementation of the networks must be undercomplete.

Deep convolutional layers and an undercomplete autoencoder architecture can be combined into a deep convolutional and undercomplete autoencoder, which will be the model

[3]There are two more compression stages in the summing of sensor data to trigger cells and in the change of datatype from 10 to 7 bit. However, these stages are finalised in the design and are therefore not further considered in this discussion.

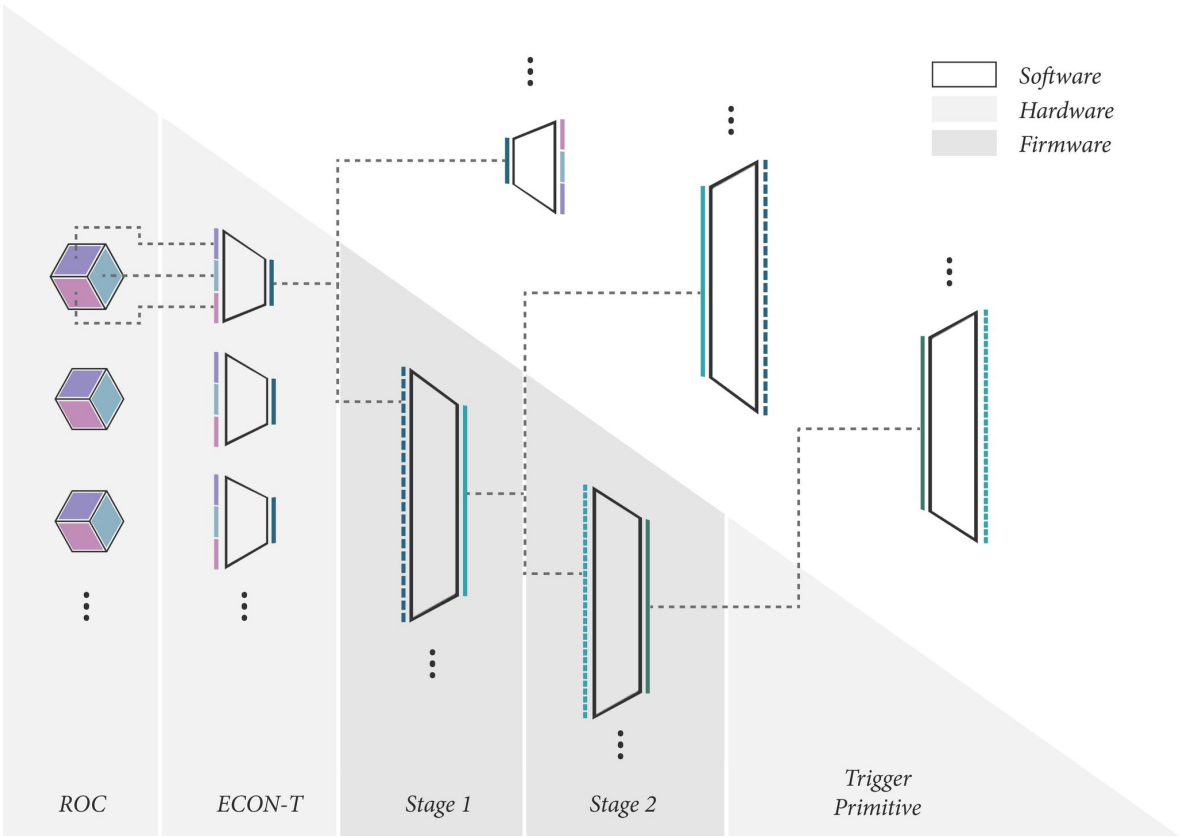


Figure 6: Illustration of a neural network based HGCAL trigger primitive generator. Trigger cell data from three ROCs is sent to one ECON-T, where the data is concatenated and encoded to the latent representation of a module. This latent space is sent together with 20 other ECON-T encodings to the Stage-1 encoder, which, again, concatenates its inputs and encodes it. Stage-1 encodings are sent together with 29 other Stage-1 encodings to Stage-2, which concatenates its inputs and encodes it. The latent space of Stage-2 is the trigger primitive. ROC and ECON-T compression will be implemented in hardware, while Stage-1 and Stage-2 is implemented in firmware. Corresponding decoders will be implemented in software in order to train the network and monitor the outputs of each stage during inference.

under investigation in this section. Keeping the real application in the ECON-T in mind, the network depth has been kept at a reasonable size to ensure implementation in limited (ASIC hardware) resources. An illustration of the architecture can be seen in Fig. 7. In order to compare the performance of neural network architectures after introducing different constraints, the saturated reconstruction (MSE) loss is used as a measure of reconstruction performance and visual fidelity of the reconstructed image.

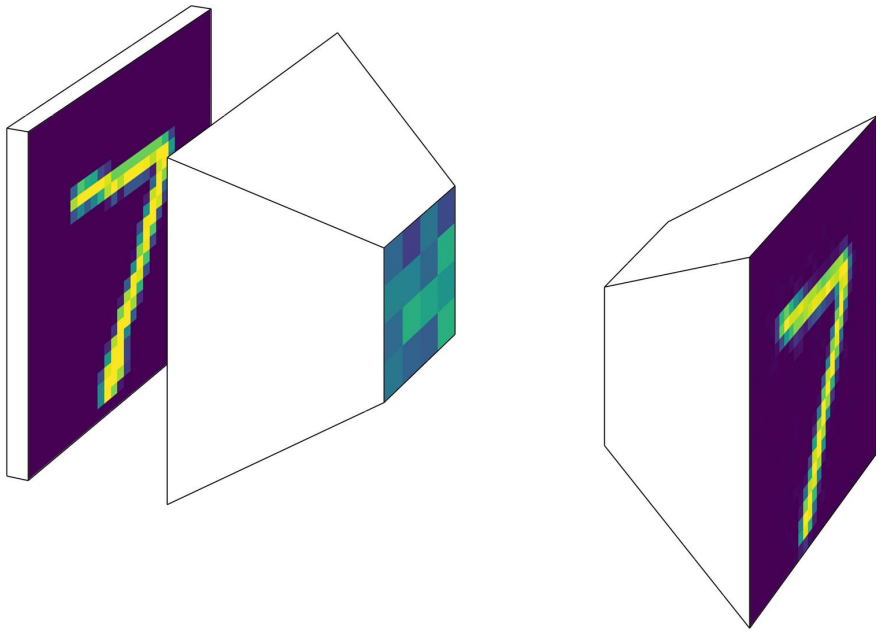


Figure 7: Illustration of convolutional autoencoder model. The encoder takes a 28×28 input image, produces a 4×4 latent space from which the decoder tries to reproduce the original image.

The basic convolutional autoencoder consists of a stack of convolutional layers, which reduce the input image dimensions with each layer, according to Eq. 4, until a sufficiently small output feature map is produced. As it is typical for convolutional layers to increase the amount of filters with depth, the resulting feature maps need to be combined at the end of the encoder in order to form the bottleneck. There are two ways to achieve this:

- By using a convolutional layer at the end of the encoder that uses a single filter and thereby collapses all preceding feature maps into a single output map, which can either be used directly as a two-dimensional latent space or flattened to one dimension. The resulting encoder structure is fully convolutional. Therefore, this model is referred to as fully convolutional autoencoder (FCAE).

- By using a fully connected (dense) layer at the end of the encoder which connects all pixels from all preceding feature maps to a one-dimensional layer of arbitrary size in order to build the bottleneck. Due to the dense layer this model produces a different latent space and is thus referred to as convolutional autoencoder (CAE). This architecture is similar to what has been so far implemented in the ECON-T.

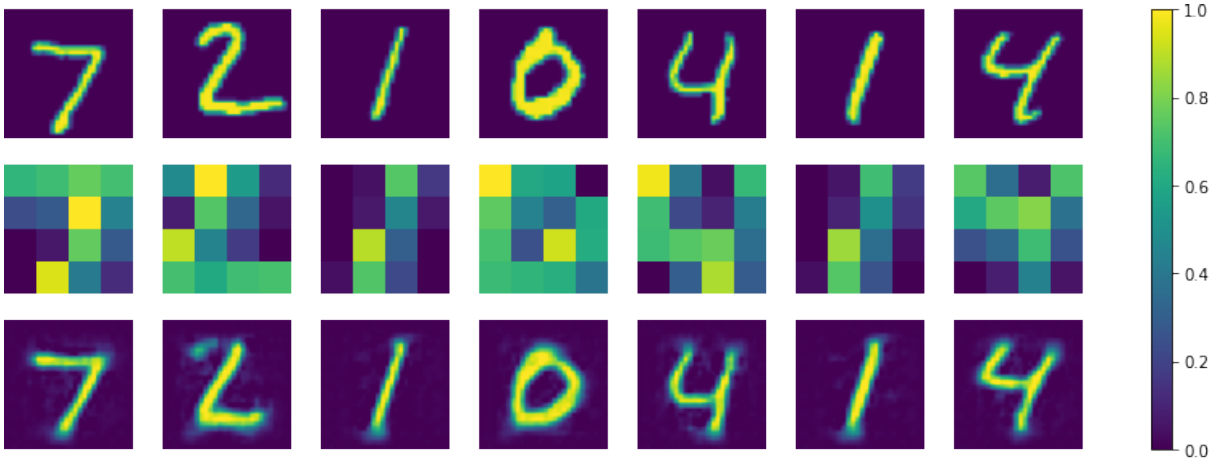
After the bottleneck, transposed convolutional layers expand the representation back into the original input space in the decoder part of the autoencoder.

While the FCAE, trained on reconstruction, produces latent representations that resemble pixelized versions of the input images, the latent space of the CAE has a more code-like nature, as seen in Fig. 8. The reason for the difference in representations can be explained by expressing the dense layer by an equivalent convolutional layer with a kernel size matching the feature map dimensions of the last encoder (convolutional) layer, no strides and an amount of channels (filters) that is equivalent to the amount of units in the dense layer. Per channel, a one pixel feature map is produced by using the same filter on all input feature maps, yielding a one-dimensional output in the channel dimension which is equivalent to the output of a dense layer. By this transformation the effect of the dense layer on the latent space can be seen more clearly – by scanning all feature maps for the same pattern, the dense (or dense-equivalent) layer looks for global features (common to all feature maps) and yields activations based on their combined presence, whereas the point-like convolutional layer combines the presence of all local features (same pixels along the channel axis) into the latent representation.

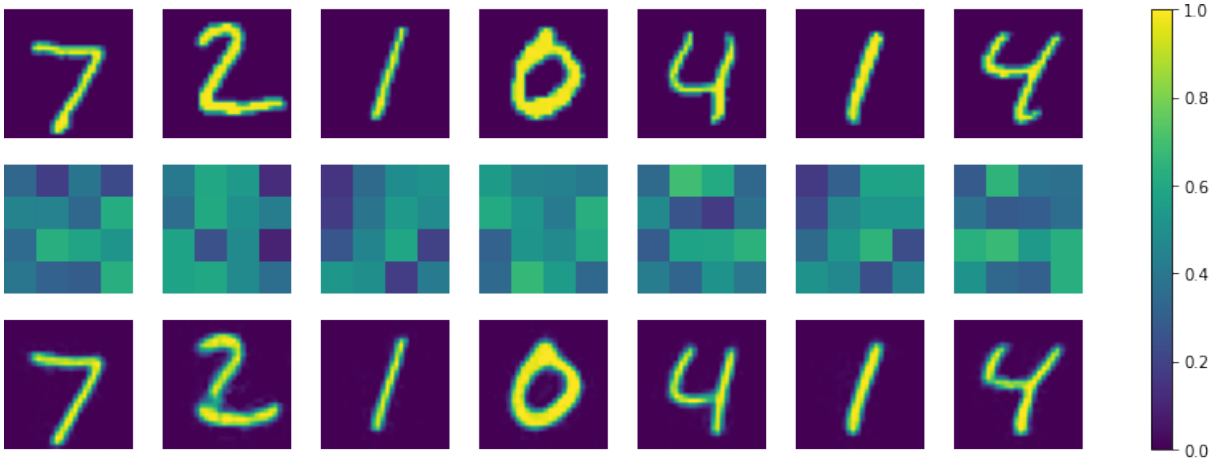
Also, the network capacity of the CAE is larger than the FCAE's, which allows modeling of more complex functions. This is confirmed by observing the reconstruction error, which is twice as large for the FCAE than when using a dense layer in the encoder and decoder. The dense layer of the decoder is not necessary to illustrate the working principle described here, however it drastically improves performance due to increased network capacity. Because the decoder in the concept is implemented in software and not subject to any size constraint, it was chosen much larger than the encoder's dense layer. Because of the superior performance of the autoencoder including dense layers and the similarity to the current implementation in the ECON-T, this architecture (CAE) was chosen as a baseline for all further studies.

A further advantage of using a dense layer in the latent space is that its dimensions can be easily changed, whereas the fully convolutional model requires significant changes to the architecture in order to produce a change in dimensions. By varying the latent dimensions while keeping the rest of the architecture unchanged, a relationship between reconstruction quality and compression ratio can be established. As reconstruction quality is not defined quantitatively, the saturated reconstruction loss on the validation set was chosen as a representative measure for information loss due to the bottleneck. An example of the progression of this loss during training for latent dimension of 16 can be seen in Fig. 9. For all latent dimensions the validation loss converges early on. However, to guarantee saturation, the best value after 1000 epochs was selected for comparison.

For different bottleneck dimensions, the loss on the validation set saturates at different values. Fig. 10 shows saturated validation losses as function of the inverse compression ratio $1/C$, bottleneck size divided by the 28×28 input dimensions of the image. If the validation loss is near zero, the reconstruction is perfect. The loss seems to approach zero already before an inverse ratio of 1 (encoder-decoder identity mapping) is achieved. The reason for this is that the handwritten digits only occupy a small subset of the 28×28 input space, thus a representation of only ~ 160 dimensions (corresponding to $1/C \approx 0.2$) is sufficient to represent all information of the digits. In other words, the current rep-



(a)



(b)

Figure 8: Examples of input images (first row), their latent space representation (middle row) and their reconstruction from the latent space (last row) for: (a) The fully convolutional autoencoder (FCAE) and (b) the convolutional autoencoder (CAE). The training progression of (b) can be seen in Fig. 9

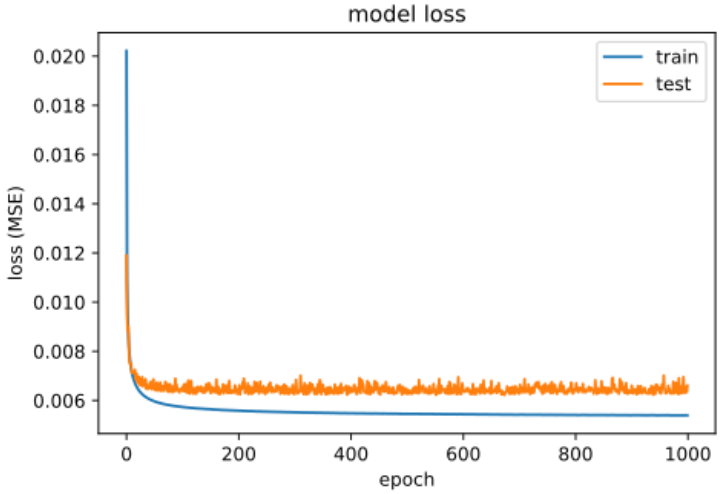


Figure 9: Progression of the MSE loss on the train set (blue) and the test/validation set (orange) for CAE with 16 latent dimensions and 1000 training epochs.

resentation of digits in 28×28 images is a rather inefficient way to store the contained information. For inverse ratios below 0.2 the network starts to lose information. However, with a $1/C \approx 0.02$ (corresponding to 16 latent dimensions) the reconstruction still yields a high visual fidelity. This ratio has been consistently used in the following studies to achieve comparability.

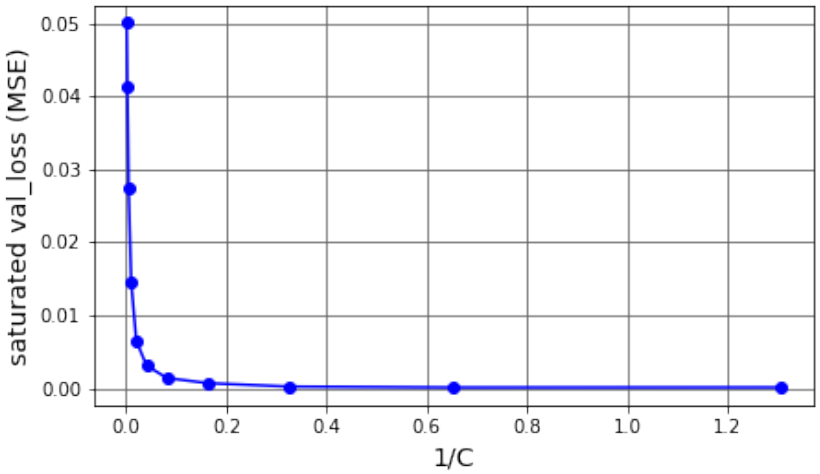


Figure 10: Reconstruction loss after 1000 training epochs as a function of inverse compression ratio of the CAE. Each data point corresponds to a five-hidden-layer deep undercomplete convolutional autoencoder trained on the MNIST handwritten digit dataset.

1.2.2 Split encoding

Shower data is usually distributed over several sensors in the calorimeter. Therefore, the encoder units only see an excerpt of the shower at any given time. Because of this, the global shower information which is desired to be preserved needs to be recovered from multiple encodings of partial information. This study investigates the feasibility and performance of such an encoding scheme. In order to compare visual fidelity with the CAE baseline model, only the structure of the encoder units was altered, while the decoder architecture was kept unchanged. This way, any changes in performance can be attributed directly to changes made to the encoder.

To achieve a limited field of view, the 28×28 input images have been split in four 14×14 patches (quadrants), which are fed in parallel to four independent encoding units that each computes a latent space. These four latent spaces are concatenated and fed to one decoder. An illustration of the architecture, referred to as *4-Split* (4S) can be seen in Fig. 11. The encoders are each composed of the same convolutional layers as the CAE encoder. However, the terminating dense layer has only four units, such that the four units together produce a latent space of 16 dimensions, equivalent to the CAE discussed in the previous section. Therefore, the dense layer at the end of each encoding unit cannot correlate global features of the entire image, but rather correlates features per quadrant. This scheme achieves the previously mentioned encoding of partial information. On the other hand, due to its individual set of network parameters, each unit has the capability to learn its own, quadrant-specific filters. The total capacity of the four units together is about 33% larger than the encoder of the baseline model (cf. Table 1). Thus, the question arises: Is the performance of the 4-Split model better because of a larger overall capacity or because of smaller dense layers and per quadrant, instead of global, correlations?

As it turns out, the 4-Split architecture (4S) performs slightly worse on the reconstruction task, despite its larger capacity. The loss saturates at a value about 20% larger than the baseline model. Examples of reconstructions and corresponding latent spaces can be seen in Fig. 12. The fully convolutional version of the 4-Split (F4S), on the other hand, performs significantly worse than the fully convolutional model (FCAE). Fig. 12a shows that the encoder splitting seems to prevent the network from building a pixelized latent representation, as seen in in the examples of the fully convolutional autoencoder (Fig 8a).

This study showed that it is feasible, without much loss in reconstruction performance to encode image patches which contain only partial information of an image, from which the global information was recovered after decoding their concatenation.

1.2.3 Invariance in ϕ

Due to the symmetry of the CMS detector, physics processes in areas of concentric rings, of $\theta = [0, 2\pi]$, around the beam axis have no preferred direction, i.e. they are invariant in ϕ. Therefore, encoder units positioned in these regions should be expected to also behave the same. For neural networks that means identical architecture and identical network parameters, which guarantees identical results at inference time. This study explores how identical units can be implemented in practice and how they compare in performance to the 4-Split with individual encoder units and to the baseline model (CAE).

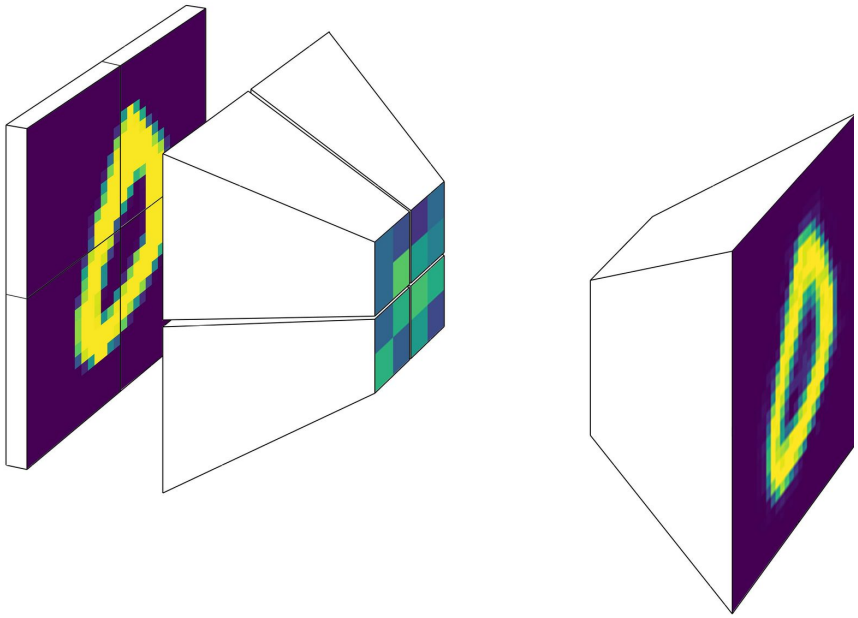
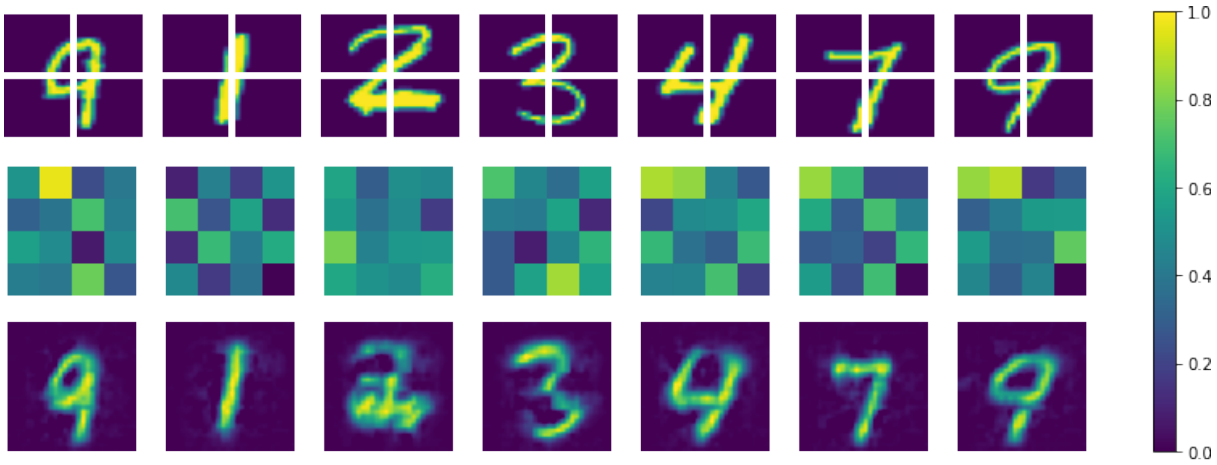


Figure 11: Illustration of 4-Split model. Four similar encoding units with individual parameters produce four 2×2 latent spaces, which are concatenated and fed to the same decoder architecture from the baseline model described in Sec. 1.2.1.

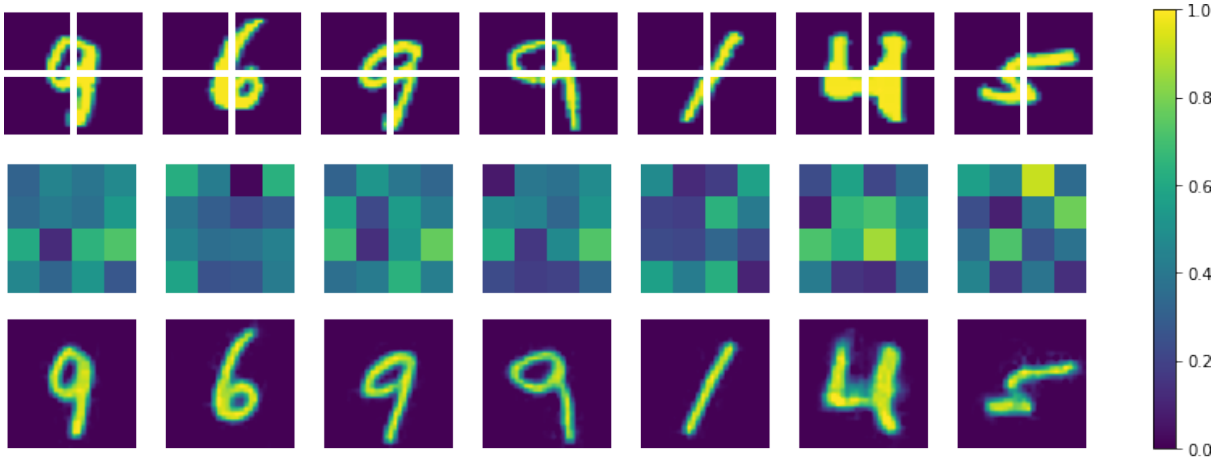
Identical units can be realized by sharing (tying) the parameters of architecturally identical encoder units. Then, the gradients of all encoder units are implicitly averaged and a shared set of parameters is updated during training time. This technique not only requires (in theory) less memory to store the parameters, but also greatly simplifies the training of the 4-Split model (4S). Instead of training four individual units in parallel, only one unit is trained sequentially on four different inputs, such that the generated latent spaces are first concatenated and then further directed to the decoder. Fig 13 shows the latent space activations of four encoder units (a) with individual and (b) with shared weights exposed to the same 14×14 input image fragments. As one can see, the latent spaces of the encoder units in (b) are identical as expected.

The reconstruction performance of the 4-Split with shared weights is slightly worse than the model with individual weights. The loss saturates at a value $\sim 8\%$ larger, which can be attributed to the much larger capacity of the model with individual parameters and its capability to learn quadrant-specific filters, whereas the one with shared weights can only draw from a common set of filters for all quadrants. However, it is surprising that the loss is not much worse, considering that the encoders with shared weights contain about three times fewer network parameters (cf. Tab. 1) than the ones with individual weights. Fig. 14 shows examples of inputs images, their corresponding latent spaces and reconstructions. In comparison to Fig. 12b, the visual fidelity does not look much worse. Only the reconstruction of the digit 5 in the last column exhibits a significantly lower level of detail.

This study showed how identical encoding units, exposed to parts of an image, can be

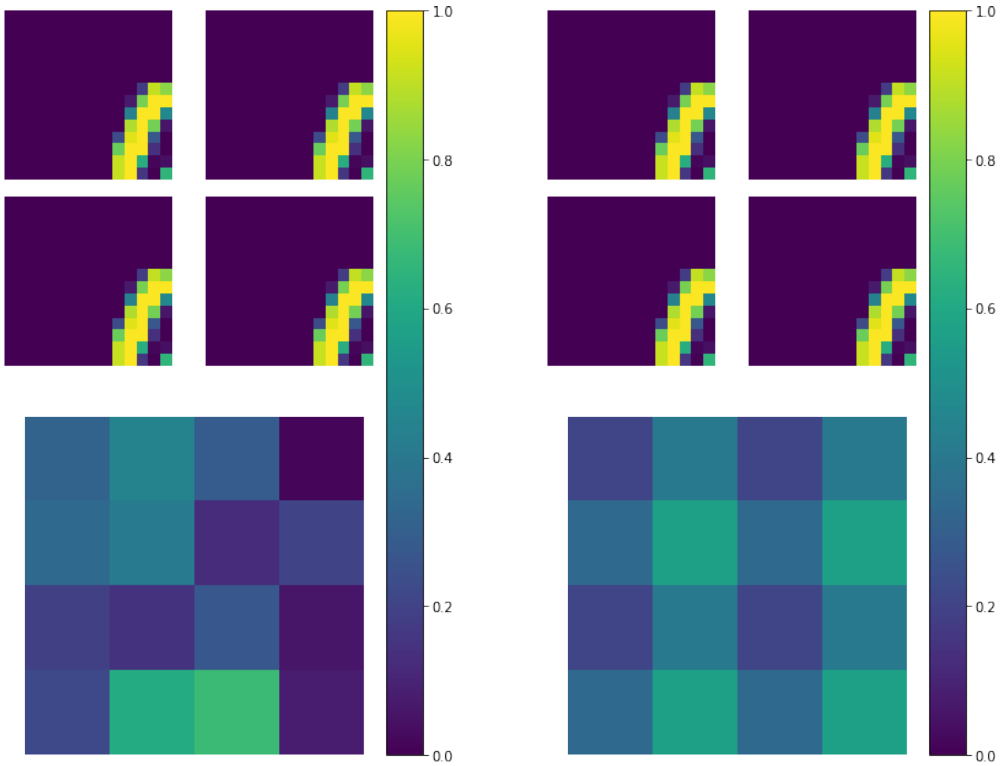


(a)



(b)

Figure 12: Examples of input images (first row), their latent space representation (middle row) and their reconstruction from latent space (last row) for: (a) Fully convolutional 4-Split (F4S) and (b) 4-Split (4S) models.



(a) (b)

Figure 13: Latent space activation for same image fragment for 4-split with (a) individual weights and (b) shared weights.

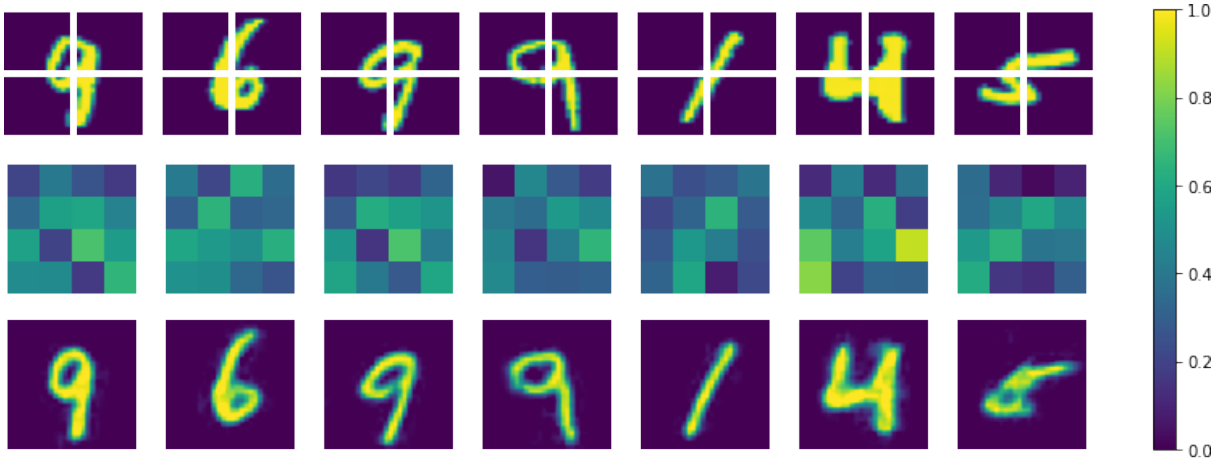


Figure 14: Examples of input images (first row), their latent space representation (middle row) and their reconstruction from latent space (last row) for the 4-Split model with shared weights.

implemented. These encoding units perform a little bit (8%) worse than encoding units with individual parameters on the reconstruction task. However, the decrease in performance is rather insignificant compared to the amount of network parameters necessary for the shared-parameter-model (3 times lower).

1.2.4 Partial encoding

In the last two sections the construction of a network architecture was discussed that encodes quadrants of an input image which are subsequently combined in order to recover the information of the full image. However, in the detector, shower data is not always distributed in equal parts on neighboring sensors. Instead, there are many possible splittings of shower data between sensors, and therefore many more encodings that need to be considered. Furthermore, since the encoder units share a common set of weights, each unit must know about the encodings for all encoders at the same time. This study focuses on simulating different shower positions, including empty sensors to test the performance under conditions that are more realistic than previously discussed.

In order to simulate different situations of sensor occupancy, the 28×28 digit is embedded in a larger 56×56 image, and translated in the x- and y-axes in a range of $0 \rightarrow 27$, such that the digit can reach all possible positions inside the larger image without moving out of its boundaries, as illustrated in Fig 15. In this way, the full information is guaranteed to be inside the input image.

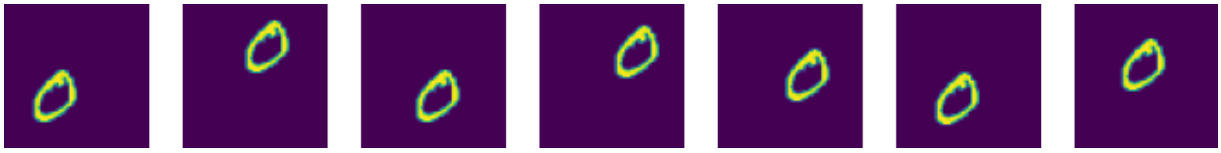


Figure 15: Example of 28×28 digits randomly positioned inside a larger 56×56 image of zeros. The digit is translated in integer steps in the range $0 \rightarrow 27$ for x- and y-axes.

By extending the 4-Split architecture with shared weights to include four times more encoder units (16 in total) and scaling up the decoder accordingly to produce 56×56 reconstructions. Each encoder covers 1/16 of the full input image and an illustration of the model architecture, referred to as 16-Split (16S), can be seen in Fig 16.

If the 16-Split is trained without shifting the digit in the larger input image, the model is very similar to the 4-Split. Due to the random positioning of the digits, the shared set of parameters needs to account for more variation in the input data, i.e. there are more 14×14 image framgments in comparison to a splitting by quadrants.

To compare the 16-Split to the other models, the reconstruction loss is evaluated only on the part of the output image which contains the 28×28 digit. Otherwise, the mean of the squared errors between output pixels and truth would pull the loss down, since most of the activations are zero. Evaluating the 16-Split's reconstruction performance leads to an increase larger than a factor of two in reconstruction loss with respect to the 4-Split with shared weights and to a 2.7$\times$ increase in comparison to the baseline (CAE) model.

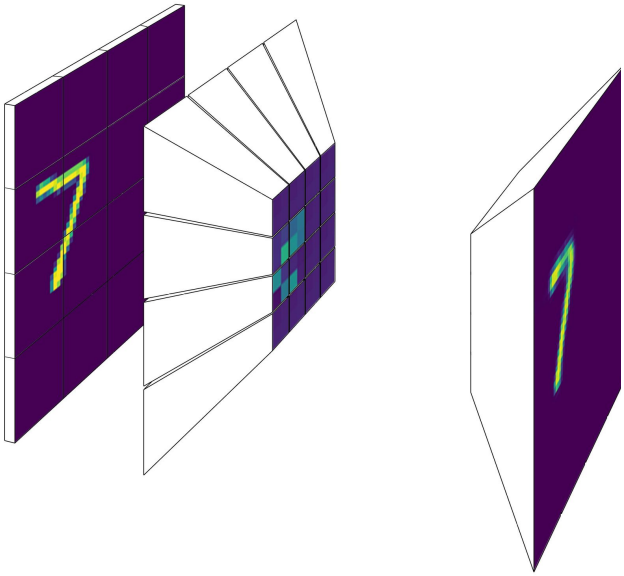


Figure 16: Illustration of the 16-Split model. The 16 encoder units have the same architecture as the 4-Split encoders, generating a four unit latent space from a 14×14 input image patch. Thus, the decoder input latent space consists of 64 (16×4) values, from which the original 56×56 image is reconstructed. The inverse compression ratio of the 16-Split is the same (0.02) as those of the 4-Split (4S) and the baseline (CAE) model.

Examples of reconstructed images and their corresponding latent space representation can be seen in Fig. 17. Since four times more encoder units are used, the latent space is composed of 16×4 encoder output latent spaces. It can be seen that most of the latent space activations are near 0, while only the encoding units that see parts of the digit in their input show an activation.

This study showed a significant decrease in reconstruction performance when the encoders need to account for different digit positions and thereby a larger variety of input patches containing part of the same global information. This is summarized in Tab. 1 which shows the validation losses for all models so far discussed after 10 000 epochs of training. The 16S model was not trained so far as it has a much larger amount of parameters and presumably requires a more careful tuning of the training process. This could be an interesting consideration for future studies.

1.2.5 Trigger primitive information

Reconstruction quality (visual fidelity), which has been discussed so far, is not the primary information needed for trigger primitives – particle type, position, and energy, on the other hand, are. Therefore, this study discusses additions to the 16-Split architecture introduced in the last section which allow to train on objectives other than reconstruction.

First, in order to classify which digit is in the image, given the combined latent spaces of the 16 encoder units, a new decoder architecture was created, referred to as classification head (16S-C). The classification head contains four dense layers with 256, 128, 64, and

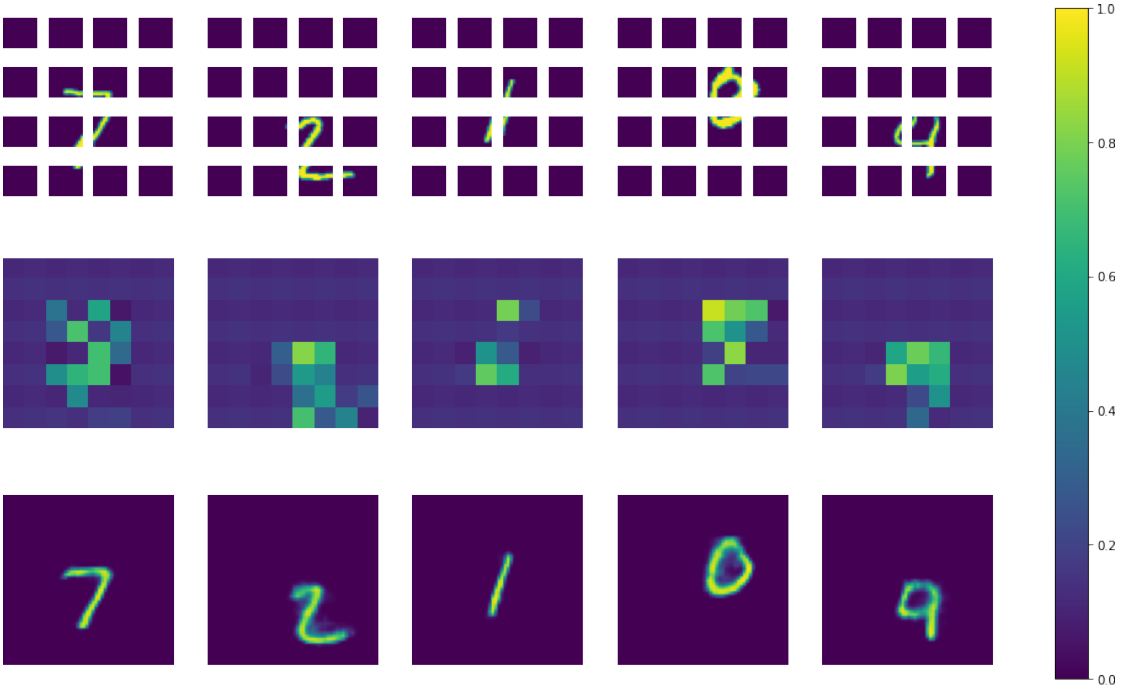


Figure 17: Examples of input images (first row), their latent space representation (middle row) and their reconstruction from latent space (last row) for the 16-Split model with shared weights.

Table 1: Training summary for different models, including number of trainable parameters for the entire model (encoder model), the training time per epoch and the saturated reconstruction loss (MSE) on the validation set after 10 000 epochs. The corresponding model architectures are detailed in Appendix **??**, and their training loss curves are shown in Appendix **??**.

Model	**Input shape**	**Parameters**	**Time per epoch**	**MSE (sat.)**
FCAE	$1 \times 28 \times 28$	39 251 (19 841)	7s	0.0156
CAE		99 097 (69 008)		0.0073
F4S	$4 \times 14 \times 14$	96 982 (77 572)	9-11s	0.0311
F4S (shared)		38 803 (19 393)		0.0332
4S		121 753 (91 664)		0.0088
4S (shared)		53 005 (22 916)		0.0092
16S	$16 \times 14 \times 14$	492 001 (366 656)	*	*
16S (shared)		148 261 (22 916)	24-26s	0.0226

10 nodes, respectively. The last layer's activations are transformed by a softmax function into a probability distribution. During training this probability distribution is compared to a one-hot vector encoding the true class corresponding to an input image using the categorical cross-entropy (CCE) loss function. In this way, the encoders and classification head are trained to minimize the difference of the last layer's activation to the true label. Fig. 18 depicts latent spaces and probability distribution outputs corresponding to different input image examples, where the class labels are distributed on the x-axis, together with the true label highlighted in blue. The representations of the encodings show a prominent checkerboard pattern. These patterns are an artifact of the strided convolutions of the encoders, which could be avoided by using a stride of one and max-pooling layers in between. Although this pattern is also visible in Fig. 17, it is noteworthy that for classification it is much more prominent, which could be an indication of a more concise latent space in comparison to the space trained on the reconstruction task. In other words, for the classification task only part of the possible capacity of the latent space is used.

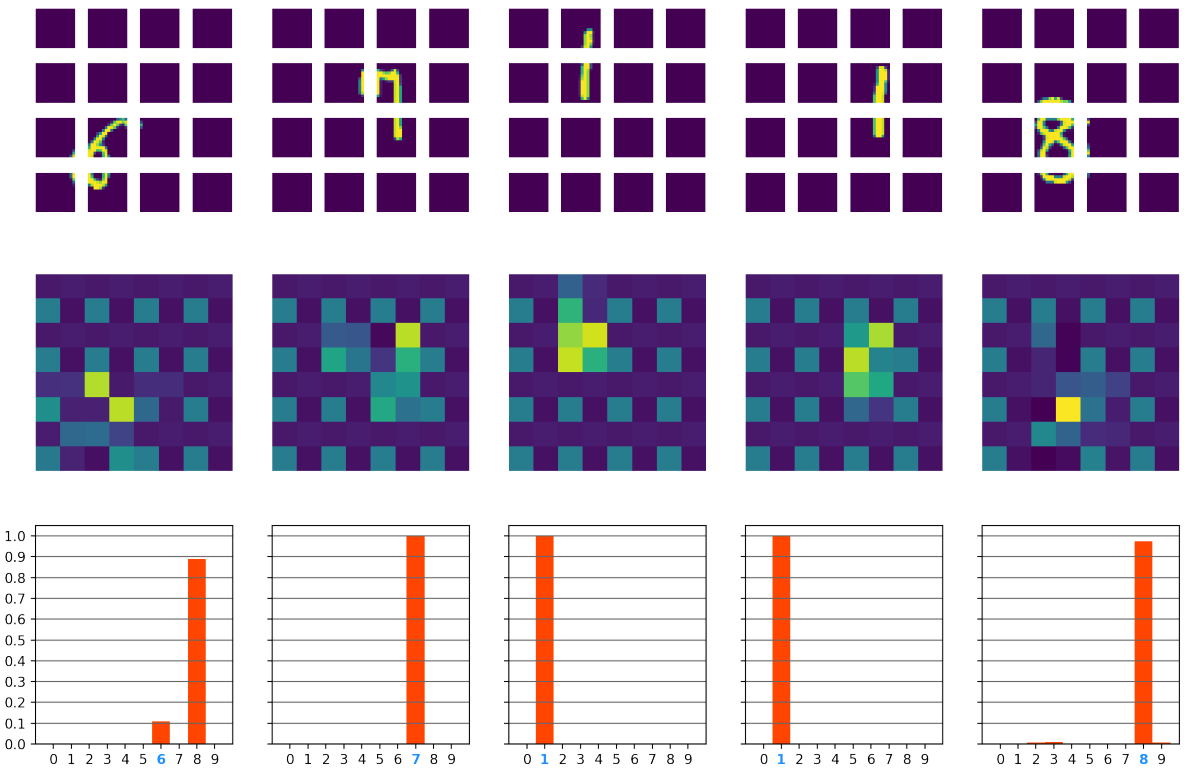


Figure 18: Examples of input images (first row), their latent space representation (middle row) and their the predicted classification (red) from latent space compared to the true class (blue) of a digit (last row) for the 16-Split with classification head (16S-C).

Second, for the purpose of regressing the position of the incident particle, given the digit image and the true position, another decoder architecture, referred to as the regression head (16S-R), was created. The regression head is very similar to the classification head – it also contains four dense layers of which the first three have the same dimensions as the 16S-C. The fourth dense layer, however, contains only two units with linear activation; for a given example with a corresponding truth consisting of a (x,y)-coordinate pair is compared to the two linear output activations via the mean absolute percentage error (MAPE). For each image the truth was defined as the center of a digit. Resulting latent spaces and regression outputs for certain example images can be seen in Fig. 19. The quantity Δr is the euclidean distance $\Delta r = \sqrt{\Delta x^2 + \Delta y^2}$ between the regression output (red circles) and the truth (blue circles). The same checkerboard pattern shows up. It seems that for the regression task even less latent capacity is needed – besides a larger contrast between the units, even units in the area of encoding seem to be mostly unused in the codes.

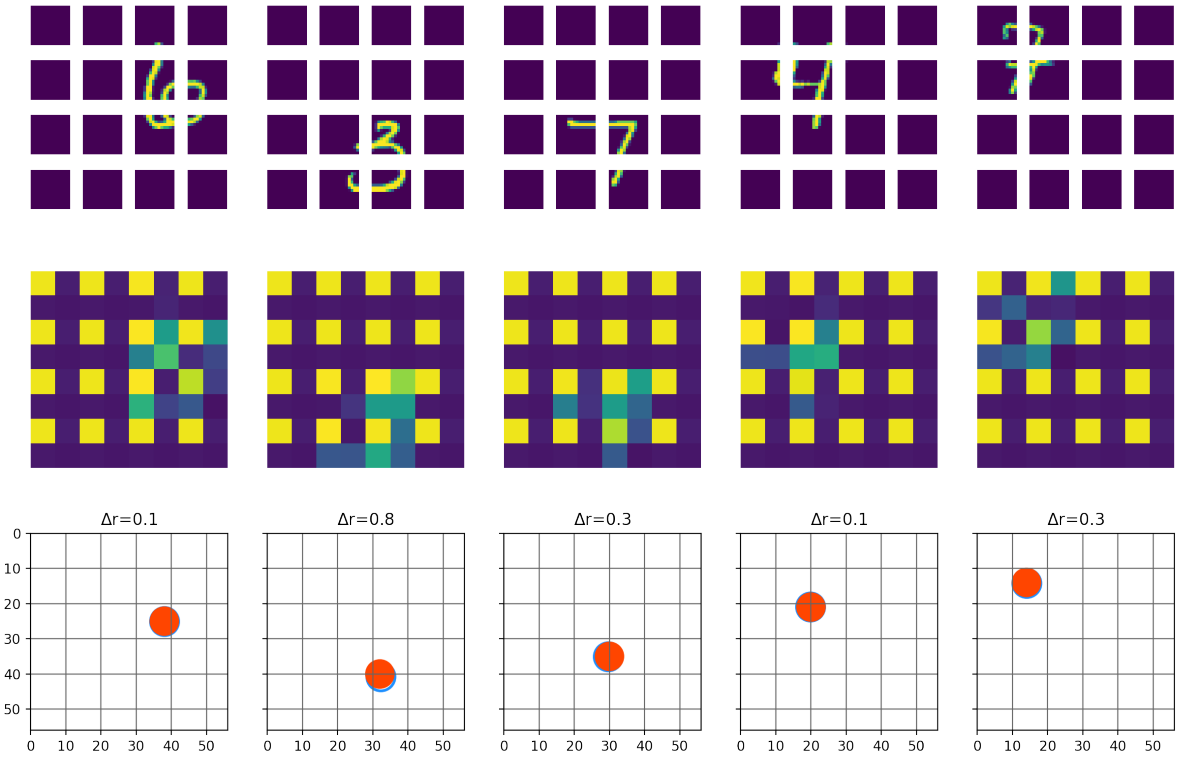


Figure 19: Examples of input images (first row), their latent space representation (middle row) and their the predicted regression of position (red circle) from latent space compared to the true position (blue circumference) of the center of a digit (last row) for the 16-Split with regression head (16S-R). Above the prediction, the euclidean distance, Δr, between truth and prediction is given.

After having seen how classification and regression can be realized within the same architecture, the next question is: How can one train the encoders in order to produce a latent representation that can be used by all decoder heads to recover different types of information from the same input image? By observing the latent representations of the classification and regression tasks one could assume that unused latent space capacity can be used in order to find such an encoding. Fig. 20 shows the latent space representation for encoding units trained on classification and regression at the same time. One can see that the contrast of the checkerboard pattern is fainter and that the latent codes seem to use more space than seen in Figs. 19 and 18. This supports the hypothesis that the pattern is related to used latent space capacity. Both heads have been trained at the same time with equal weight on the loss function. As it turns out, training with several objectives always involves a trade-off, with best possible performance only in case a head is trained in isolation. This can be seen more clearly in Fig. 21. It illustrates how the saturated loss for a certain head varies with its loss weight LW on the global loss $\mathscr{L}_{glob} = LW_{class} \cdot \mathscr{L}_{CCE} + LW_{reg} \cdot \mathscr{L}_{MAPE}$ used to update the network parameters. Tab. 2 shows an overview of training reconstruction (MSE), classification (CCE) and regression (MAPE) heads in different order, fixing the encoder weights and training the other remaining heads on the (fixed) encoder. It shows that the best performance can be achieved when the encoder is trained for a single task. The second best performance can be observed when all heads are trained simultaneously. However, the loss weights, which specify the impact of a certain task on the total loss, are hyper-parameters that would need to be fine-tuned in order to maximize the performance. Furthermore, this study yielded several other insights into the training processes:

- **Row 1 vs. row 5** shows a significant decrease in classification accuracy when the classifier is trained on the encoder pretrained for reconstruction and regression. If the encoder is pretrained on reconstruction alone, the classifier can be trained (together with the regressor) to yield significantly better performance. The representations for reconstruction must be more favorable for classification than the representations for regression and reconstruction together.

- **Row 1 vs. row 6** compares encoders pretrained on reconstruction to encoders pretrained on reconstruction and classification. The latter seems to generate representations that are unfavorable for the regressor to learn a meaningful mapping. This is a similar behavior to the first point, as in both cases the task trained on pretraining with two other tasks looks significantly worse. However, this cannot be generalized since row four shows not significantly worse reconstruction performance when the encoder was pretrained on classification and regression.

- **Row 3 vs. row 7** shows a surprising improvement of MAPE loss when all objectives are trained together. During training it was noteworthy that the MAPE loss was largely fluctuating. It might be, that the model was not trained long enough and that the MAPE value would, after more training, reach a better value for the training on single task, which would be expected. It can also be seen in Tab. 3 that the MAPE loss does not always behave as expected. Overall, the training of all heads at the same time yields the best performance.

This study showed the performance on the tasks relevant for trigger primitive generation considering the most significant architectural constraints imposed by the real-world system. Noise (similar to pile-up) has not been discussed so far, which would be interesting

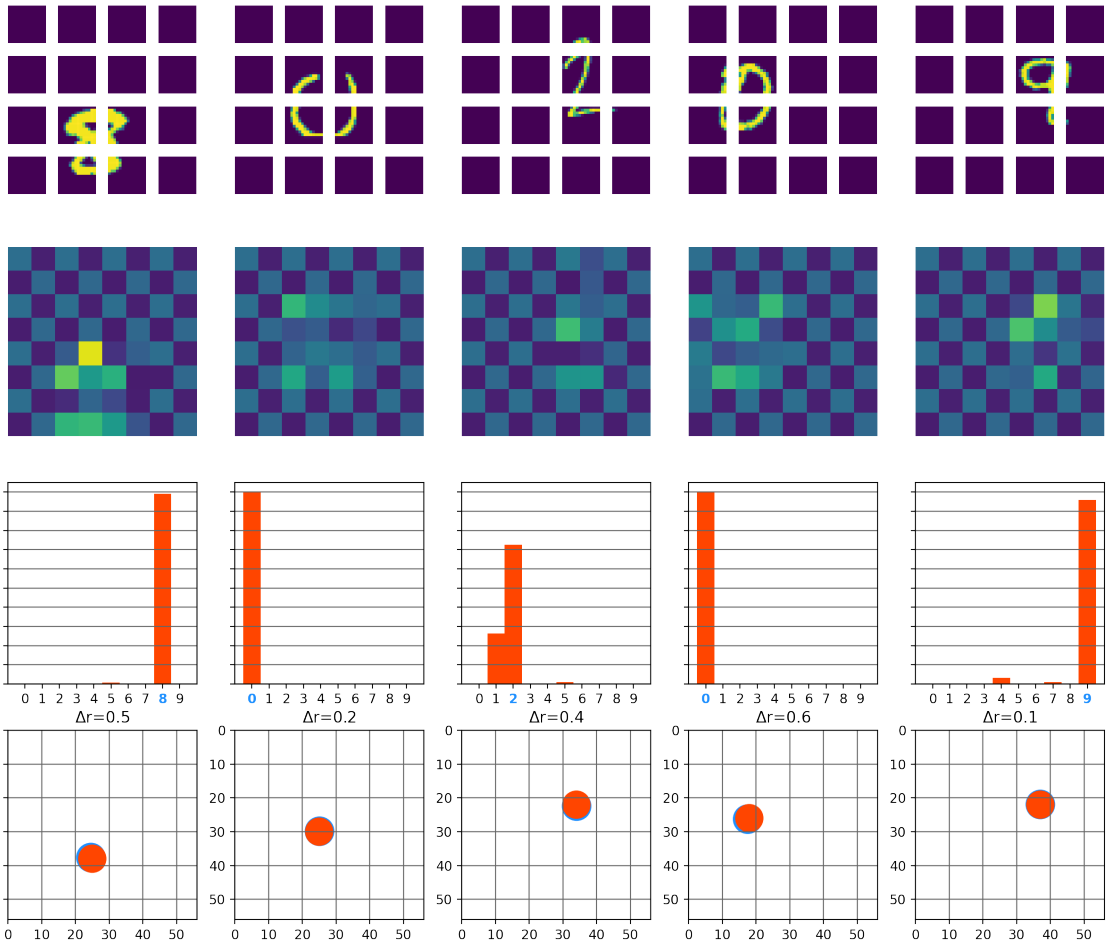


Figure 20: Examples for 16-Split trained with classification and regression heads at the same time. Examples (first row), latent spaces (second row), classification probability distribution (third row), and regression results (last row).

Table 2: Comparison of performance for different ways to train the 16-Split model for three different types of objectives: reconstruction (MSE loss), classification (CCE loss, binary accuracy metric), and regression (MAPE loss). The tasks without brackets in a row have been trained first, after which the encoder parameters were not changed anymore during training. The task in brackets comes from training the decoder on this (fixed) encoder. When more than one objective is trained at the same time, relative loss weights LW had to be specified which have been chosen somewhat arbitrarily, and could probably be tuned to get better performance.

Encoder loss weights			**Decoder loss performance**		
MSE	**CCE**	**MAPE**	**MSE**	**CCE, Acc.**	**MAPE**
1	(1)	(1)	0.0226	(0.5811, 83.29%)	(1.5123)
(1)	1	(1)	(0.0494)	0.3248, 92.66%	(2.9215)
(1)	(1)	1	(0.0352)	(0.8090, 73.13%)	1.2721
(1)	**1**	**1**	**(0.0302)**	**0.3986, 88.49%**	**1.5217**
1	(1)	0.01	0.0269	(2.3115, 9.4%)	1.2870
1	0.05	(1)	0.0249	0.4767, 88.59%	(27.4358)
1	0.05	0.1	0.0340	0.5136, 83.18%	1.2143

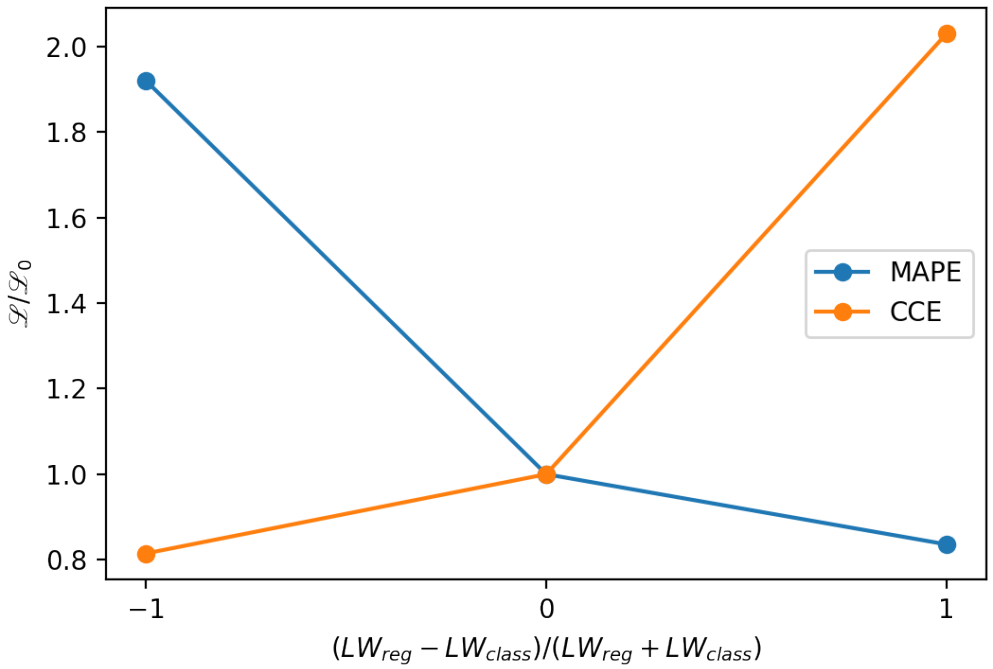


Figure 21: Asymmetric progression of two different losses (MAPE, CCE) as function of their contribution (encoder loss weight LW) to the global loss. A value of -1 corresponds to pure classification, $+1$ to pure regression.

to explore in further studies. The denoising application of autoencoders encourages such studies.

1.2.6 Information bottleneck

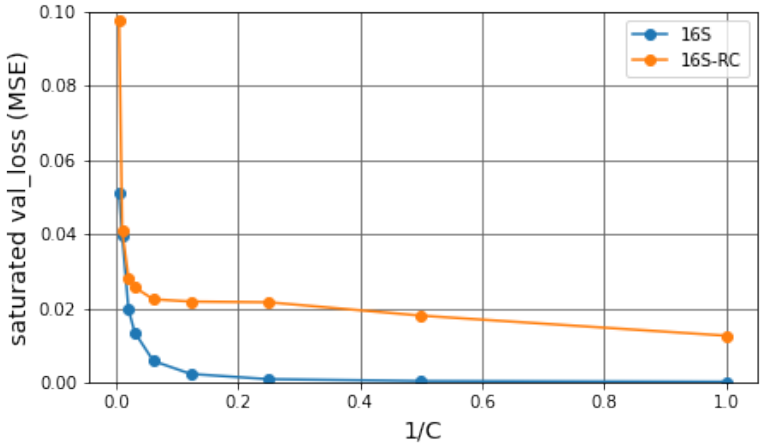
In the last experiment, the prediction performance of the three objectives with respect to the size of the latent space was investigated. For this purpose, the dense layer of the 16-Split, that defines the size of the latent space, has been varied while the rest of the architecture was kept unchanged. The saturated loss values as a function of the inverse compression ratio between input and latent dimensions was determined and can be seen in Fig. 22.

It can be seen that the losses on the validation set follow the same falling exponential trend as the CAE (Fig. 10) when changing the bottleneck size. Saturation seems to take place at a relatively early stage ($1/C = 0.02$), suggesting that already small amount latent dimensions satisfy all three objectives reasonably well, and that further increase of size does not increase the performance much, without changing the training strategy or augmenting the datasets.

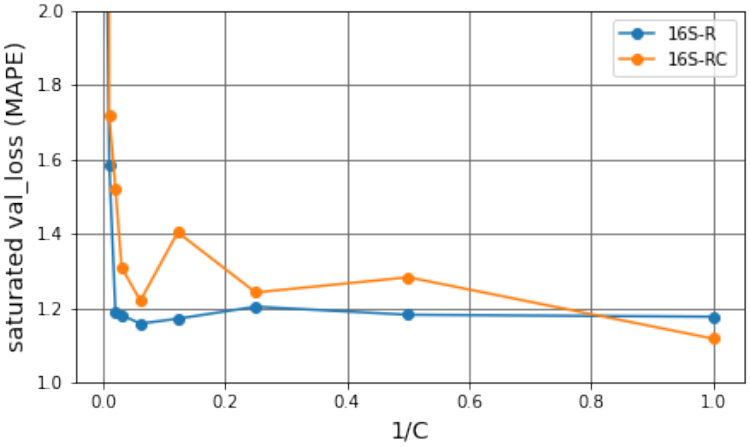
Tab. 3 shows the corresponding measurements. Classification accuracy and regression loss seem to saturate very early on, while reconstruction loss is still improving. This supports the hypothesis made in the previous section – classification and regression tasks need less latent capacity and therefore saturate earlier. The reason why they are still (slightly) improving can be linked to a larger dense layer per encoder, thus a more informative code per image fragment to compensate for the random positioning of the input digit.

Table 3: Loss performance of different objectives after varying the inverse compression ratio $1/C$ (latent dimensions/input dimensions) of a model while keeping the rest of the architecture unchanged. Models are trained until saturation on either reconstruction, classification or regression alone (16S, 16S-C, 16S-R) or the encoders are trained on regression and classification, after which the encoder parameters are fixed and the reconstruction head is trained on top of this (16-RC). For each model the values corresponding to best classification accuracy are selected and summarized in the table. Models with larger capacity needed to be trained with an exponentially larger number of epochs in order to reach saturation level.

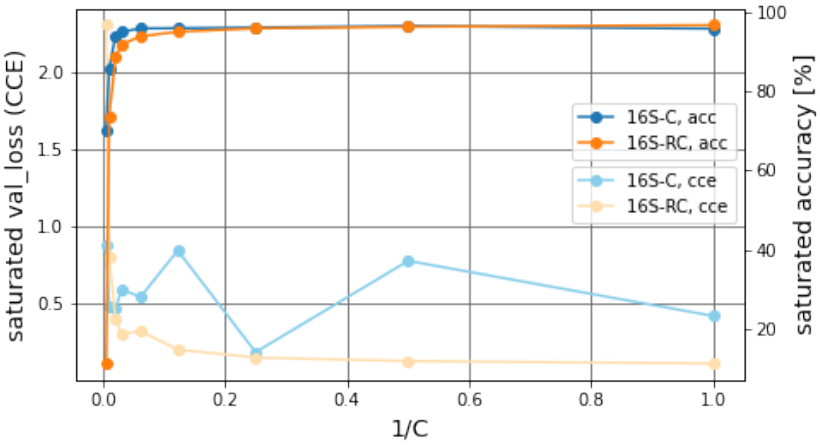
1/C	**MSE**		**CCE, Acc.**		**MAPE**	
	16S	**16S-RC**	**16S-C**	**16S-RC**	**16S-R**	**16S-RC**
0.005	0.0509	0.0976	0.8782, 70.15%	2.3011, 11.34%	2.3214	28.2130
0.010	0.0394	0.0409	0.4783, 85.79%	0.8049, 73.68%	1.5861	1.7168
0.020	0.0197	0.0281	0.4653, 93.88%	0.3986, 88.49%	1.1892	1.5217
0.030	0.0135	0.0256	0.5932, 94.88%	0.3026, 91.75%	1.1805	1.3084
0.061	0.0058	0.0224	0.5422, 95.76%	0.3224, 93.76%	1.1580	1.2194
0.122	0.0023	0.0218	0.8431, 95.80%	0.2026, 94.91%	1.1710	1.4032
0.250	0.0009	0.0216	0.1859, 95.99%	0.1509, 95.77%	1.2034	1.2416
0.500	0.0004	0.0180	0.7766, 96.37%	0.1284, 96.17%	1.1817	1.2824
1.000	0.0002	0.0126	0.4204, 95.70%	0.1129, 96.60%	1.1764	1.1173



(a)



(b)



(c)

Figure 22: Saturated loss value as a function of decoder input latent dimensions for the 16-Split encoder model trained with different heads and with different learning objectives. The curves correspond to the values summarized in Tab. 3.

This study showed that above a certain threshold of latent dimensions, the performance on all tasks does not improve much. Classification seems to saturate earlier on in performance. Regression of total energy, as the third objective of trigger primitive information, was not discussed so far, which should be included in future studies.

1.3 Discussion

In summary, the six studies above show the feasibility and performance of a neural network for the purpose of trigger primitive generation by considering major architectural detector properties and constraints.

The theoretical information content desired to preserve for trigger primitive generation amounts to 131 bits: three bits for the type, three 32 bit floats for the position, and one 32 bit float for the energy of the shower-inducing particle. Fig. 23 shows test beam data for a shower caused by a 250 GeV electron in eight consecutive modules convering $27\,X_0$.

In the final system, eight (low-density) modules will mount 8×3 ROCs, each accessing 16 trigger cells with 7 bits per cell which gives an input space of 2688 bits. If one only considers 99 bits for predicting the particle class and its position, an inverse compression ratio of $99/2688 \approx 0.037$ should theoretically be sufficient to recover the information of this particular shower.

As the studies in Sec. 1.2 showed, a neural network trained on classification and regression can recover the desired information on a similar toy problem with fairly accurate precision. Fig. 24 shows classification accuracies per digit class on the entire validation set in a confusion matrix for a 16-Split model of 128 latent dimensions. The compression ratio between input space and bottleneck of this model amounts to 0.04, comparable to the ratio of 0.037 for the example shower above. It is noteworthy that the toy dataset contains about twice as many classes as the real shower data.

The true positives (along the diagonal of the matrix) show a maximum difference of 14%. Although this result cannot be directly compared to the performance of the ECON-T algorithms in Fig. 5, there exists a parallel between the increase in trigger rate at constant offline threshold and classification accuracy of a neural network based approach. This parallel could allow to answer the question as to whether the neural network performance is better on different particle types than the current ECON-T algorithms and is therefore worth exploring in future studies.

Fig. 25 shows the median residuals inside upper and lower quartiles for regression of position in the x- and y-coordinates. The closeness of the median residuals to zero show that there is no pathologic deviation between the regressed value and the true coordinate.

Furthermore, the impact of pile-up in the detector has not been considered so far. Autoencoders are often used for the purpose of noise reduction, i.e. to produce a noise-free reconstruction from a noisy input image. The performance on classification and regression could be determined in a future study. Furthermore, these studies considered only a single stage of encoding, whereas the proposed concept is composed of three such stages. The effect of chaining multiple stages should also be subject of a future study.

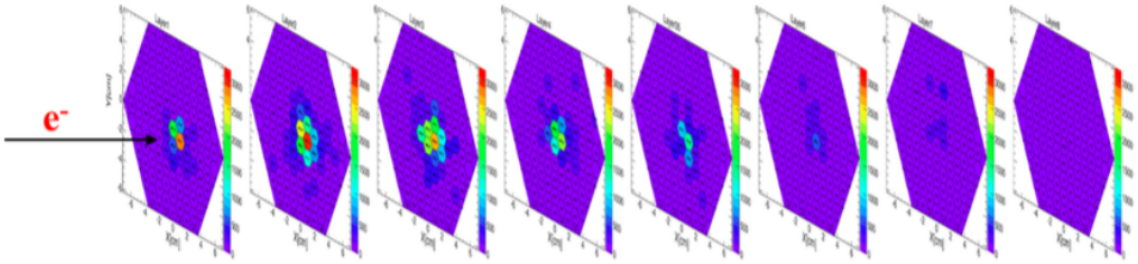


Figure 23: Example of test beam data for sensor occupancy due to a shower induced by a 250 GeV electron [**?**].

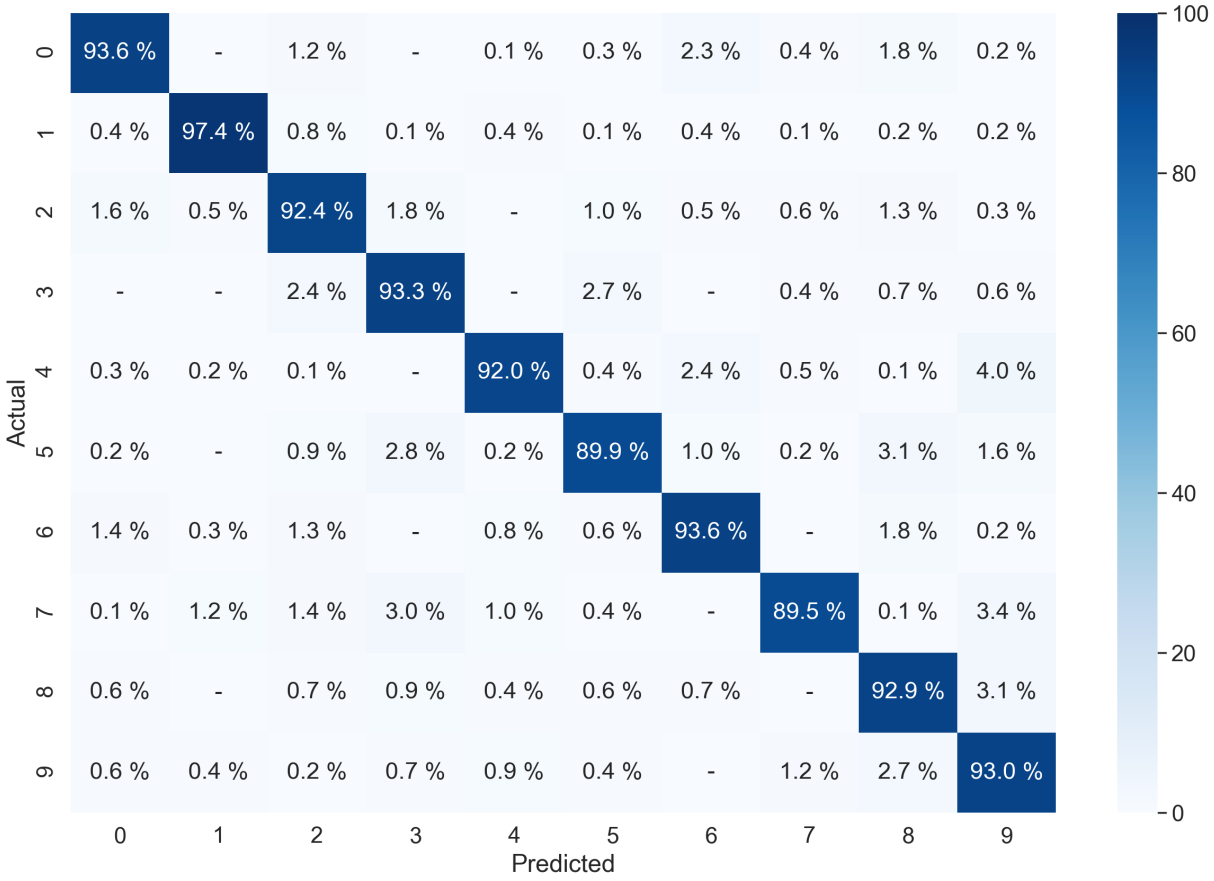
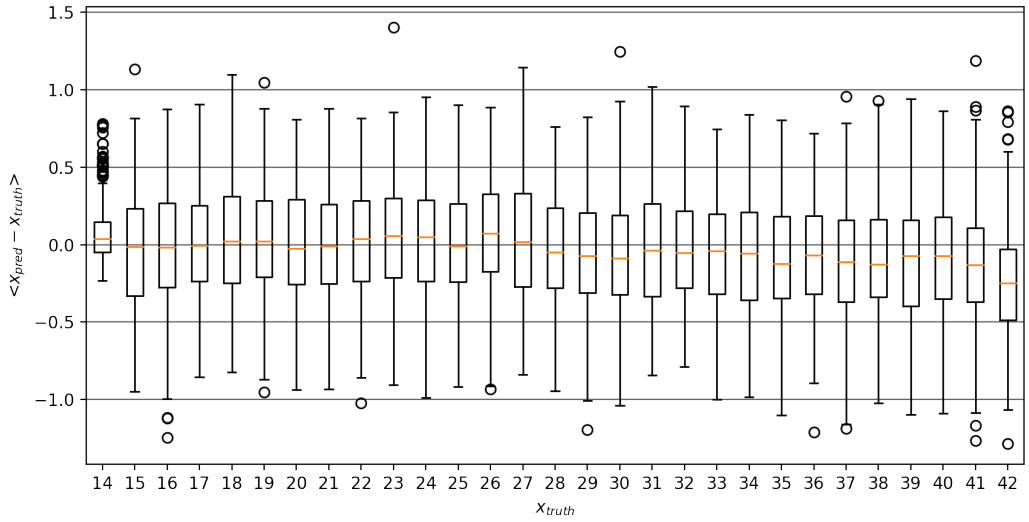
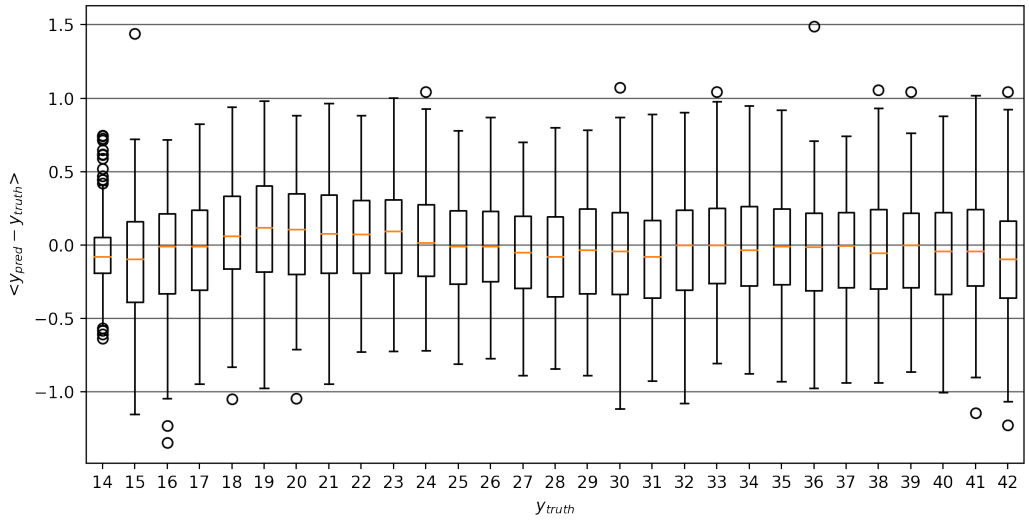


Figure 24: Confusion matrix showing prediction accuracy of true positives along the diagonal and accuracy of false positives in all other positions. The plot summarizes the classification results on the validation set for 16-split trained on regression and classification simultaneously. The x-axis shows the predicted digit (class) labels and the y-axis the actual labels.



(a)



(b)

Figure 25: Box plot showing the median (orange) in its interquartile range (box) and outliers are marked by circles for 16-Split (trained on regression and classification simultaneously, and with a latent space of 128 decoder input dimensions) regression on the validation set: (a) The x-coordinate and (b) the y-coordinate. The positions are calculated from the center of the 28×28 digit embedded in the 56×56 image and cover the range (14,14) to (42, 42).