Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Masterarbeit Informatik

Semi-supervised Learning with Locally Embedded Autoencoder

Sebastian Penhouët

31. Juli 2019

Gutachter

Prof. Dr. Martin Butz
Cognitive Modeling
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Prof. Dr. Hendrik Lensch
Computer Graphics
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Dr. Sebastian Otte
Cognitive Modeling
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Penhouët, Sebastian:
Semi-supervised Learning with Locally Embedded Autoencoder
Masterarbeit Informatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 31.01.2019 – 31.07.2019

Abstract

Our world can be described as a hierarchical structure of interrelated discriminative objects. Due to limited human resources, datasets labeled at this level of detail are non-existent. This work presents a novel operation that could help to extract local explanatory factors and their interrelations. This operation consists of a locally embedded autoencoder using recent techniques to encourage disentanglement. This locally embedded autoencoder is combined with a supervised to a semi-supervised learning task. For the proposed operation an analysis is performed that contains a comparison against max-pooling, a test of the synergy between reconstruction and supervised objective, and an investigation into the effects on sample complexity, robustness against adversarial attacks, and equivariance to spatial transformations. The learned reduction function outperforms max-pooling in terms of forwarding task-relevant information. Evidence suggests a decreased sample complexity. Empirical results show that convolutional neural networks are equivariant to translation and scale transformations until the last layer with this work also providing equivariance to orientation transformations. While a stronger emphasis on reconstruction performance does not correlate with improved supervised performance, both objectives can be pursued without negative impact.

Kurzfassung

Unsere Welt kann als hierarchische Struktur aus unterschiedlichen zusammenhängenden Objekten betrachtet werden. Aufgrund des hohen Arbeitsaufwands existiert kein Datensatz mit einem solchen Detailgrad. Mit dieser Arbeit wird eine neuartige Operation präsentiert, die dabei helfen könnte lokale beschreibende Faktoren und deren Zusammenhang zu extrahieren. Diese Operation ist ein lokal eingebetteter Autoencoder, der die neuesten Techniken verwendet, um eine Trennung der Kodierung in ihre zugrunde liegenden beschreibenden Faktoren zu erzielen. Der lokal eingebettete Autoencoder wird mit einer überwachten Aufgabe kombiniert und ist daher dem semi-überwachten Lernen zuzuordnen. In Bezug auf die vorgestellte Operation wird ein Vergleich zur Max-Pooling Operation gezogen, das Zusammenspiel zwischen Rekonstruktionsziel und Klassifikationsziel untersucht und analysiert, wie äquivariant zu räumlichen Transformationen, wie dateneffizient und wie robust gegen Adversarial Attacks diese ist. Die erlernte Reduktionsfunktion can wichtige Informationen besser weiterleiten als Max-Pooling. Es gibt Hinweise darauf, dass diese Methode zu einer höheren Dateneffizienz führt. Nach den empirischen Ergebnissen sind Convolutional Neural Networks bis auf die letzte Schicht äquivariant zu Verschiebungen und zu Skalierungen. Die in dieser Arbeit vorgestellte Operation ist darüber hinaus auch äquivariant zu Rotationen. Eine höhere Gewichtung der Rekonstruierbarkeit hängt nicht mit einer besseren Klassifikationsperformanz zusammen. Es ist möglich beide Ziele gleichermaßen zu optimieren, ohne eines von beiden negativ zu beeinflussen.

Acknowledgments

My deepest appreciation goes to Sebastian Otte. Without his continuous guidance and encouragement, this thesis would not have been possible. Our many insightful discussions helped me a lot to spark new ideas. I am particularly grateful for his academic advice.

I would like to thank Martin Butz and Hendrik Lensch for their insightful comments and suggestions.

I appreciate the feedback offered by Wieland Brendel on the focus and the overall theme of this work. This led me to question some initial ideas and finally to a more focused approach.

I thank Maximus Mutschler for his support for the TCML cluster.

Special thanks to Matthias Rieger for his review and feedback on this work.

I owe a very important debt to my girlfriend Anja Rothweiler, who provided me with moral and emotional support.

Contents

Chapter 1

Introduction

Visual processing in our brain works in stages with every stage grouping features into higher-order objects (Ward, 2015). Every level in this hierarchy could be defined as a discriminative representation. In computer vision, object recognition is concerned with computer algorithms that perform classification, localization and detection of objects within digital images. We humans are very good at detecting objects within images. We can mentally cut an object out of a 2D image and rotate it as a coherent 3D model (Shepard and Metzler, 1971). If we perceive such a hierarchical structure, this is no surprise as we know all subobjects, their properties, and their spatial interrelation. Artificial neural networks (ANNs) are machine learning (ML) techniques that currently are the state-of-the-art in object recognition. Their performance enables computers to perform tasks that were previously only solvable by humans. ML techniques approximate functions by examining a vast amount of examples. Convolutional neural networks (CNNs) are currently the most popular ANN variant in object recognition. Technically speaking, CNNs do have multiple stages of representations but these are highly entangled without having explicit hierarchies or explicit discriminative factors. Object recognition tasks generally only focus on the discrimination of objects from a hierarchy layer. There is no information on discriminative factors for the intermediate representations. This is mostly because example data has to be provided by humans and that the sheer number of necessary examples makes an annotation at this level of detail impossible. A technique is needed that learns to disentangle these intermediate representations into their explanatory factors without examples (Bengio *et al.*, 2013). To preserve the spatial interrelation between explanatory factors, it might be important to preserve spatial information in these intermediate representations (Hinton *et al.*, 2011).

This work presents the locally embedded autoencoder (LEA), a novel operation that supports learning a hierarchy of explanatory factors. It uses recent techniques from the field of unsupervised learning to learn meaningful representations without examples. Especially recent autoencoder (AE) techniques that encourage the disentanglement of representations into their explanatory factors are used. The operation uses windowing and stride, which are common techniques in CNNs, to focus on objects of different hierarchical levels. The reconstruction objective of AEs is used to preserve spatial information by obtaining equivariance to spatial transfor-

mations. At its core, this work is about representation learning. It is completed by analyzing the usefulness of these representations for a supervised objective. Both objectives are combined in a semi-supervised learning setting.

1.1 Background

Bengio *et al.* (2013) advocate for the importance of the disentanglement of intermediate representations into explanatory factors. This perspective highly correlates with the motivation behind this work. To tackle this idea, recent AE techniques that encourage disentangled representations are used. Kosiorek *et al.* (2019) emphasized the importance of spatial interrelations for object recognition. This matches the notion of a hierarchical composition of objects and their spatial interrelation (Hinton *et al.*, 2011). In this work, the notion of spatial interrelations is approached by preserving spatial information through the reconstruction objective. The notion of hierarchical objects is addressed by local views. These local views are the result of a windowing operation.

In the beginning, AEs were used as an unsupervised preprocessing method (Ballard, 1987; Hinton, 2006). This did prove to be a successful method to reduce the dimensionality without losing essential information for subsequent supervised tasks. Instead of applying AEs only as a preliminary method to a supervised task, this work integrates AEs in the training process of supervised tasks.

Supporting supervised with unsupervised learning is a longstanding research topic. Existing semi-supervised learning approaches implement a joint training of supervised and autoencoder objectives (Ranzato and Szummer, 2008; Valpola, 2015; Rasmus *et al.*, 2015; Zhang *et al.*, 2016). They either used a network of stacked AEs or added AEs onto an existing supervised learning model as an auxiliary objective. Although these methods were successful at improving supervised tasks by adding an encoding objective, they did not enforce an explicit hierarchical structure of explanatory factors. This work extends on these approaches by encouraging disentanglement within the codes, similar to (Sønderby *et al.*, 2016; Siddharth *et al.*, 2017). Additionally, this work enforces a hierarchical structure by encoding local views and focuses on a more strict separation between both objectives by using alternate training.

1.2 Objective

The objective of this work is subdivided into four parallel phases, design, implementation, exploration, and analysis. In the design phase, an operation was conceptualized that supports an image-based supervised learning task by learning to extract local explanatory factors and their interrelations in an unsupervised learn-

ing manner. The operation was designed for a parallel training of both objectives. In the implementation phase, the operation was implemented using a popular ML framework and the respective programming language. Integrability and flexibility were important requirements for this implementation. Integrability hereby refers to the integration into existing architectures or more generally an easy usage within the current paradigm of the respective ML framework. Flexibility hereby refers to the standardization of internal methods so that these can be quickly swapped to easily perform experiments. In the exploration phase, different ideas and concepts were tested in short and non-exhaustive experiments to get a feeling for internal workings. The experience and knowledge gained from these explorations guided the implementation and analysis phase. Within the analysis phase, the operation was evaluated and tested for potential improvements and underlying hypotheses. For a sound analysis, this work defined appropriate metrics, visualizations, baselines, datasets, and tests. The tests showed the capabilities regarding unsupervised learning of local explanatory factors. One test evaluated if spatial information is preserved. Since the used AEs perform a reduction, this operation was directly compared against a pooling operation. Another test examined potential improvements in sample complexity and robustness. A final test investigated the effect of this operation on the associated supervised learning task.

1.3 Outline

This work starts with Chapter 2, explaining techniques this work builds upon and related terms.

Chapter 3 introduces the LEA operation and the considerations that went into different decisions. The main points are the implementation of hierarchical levels, the integrability of the operation, and the choice between different autoencoder types.

Chapter 4 lays out how the research for this work was conducted and how the quality of reported results and findings is ensured. For this, it introduces different metrics and visualization that were used during the execution of experiments and analyses. These metrics and visualizations are also used to display results and findings.

Chapter 5 gives insight into the LEA and elaborates on important results of the various conducted experiments. This chapter tests disentanglement with AEs and the discrimination of local explanatory factors. It examines if the reconstruction objective helps to preserve spatial information. The loss in information for different window sizes is determined and compared against standard max-pooling. The performance on dataset subsets of different sizes is evaluated to measure the sample complexity. This chapter also takes a look into the robustness against adversarial attacks. Finally, the effect of the usage as auxiliary regularization on a supervised

learning task is analyzed.

In Chapter 6, downsides to the proposed operation and the presented results are identified.

Finally, Chapter 7 completes this work by summarizing the results and pointing out open questions and potential future research.

Chapter 2

Foundations

This chapter provides a shallow introduction to ML with a focus on artificial neural networks (ANNs). This introduction is followed by an extended look at ANN techniques. Finally, an explanation of spatial transformations and representational properties is given.

2.1 Machine Learning

ML tries to solve problems by looking at examples. The goal is to achieve generalization that is the ability to solve the problem for unseen data. Given there exists a function f that can solve this problem, then the examples can be considered as samples of this function. The goal is to approximate f by using these samples as references. Given a perfect approximation, f should output the correct result for an input x. The samples used during this approximation are called training data. The unseen samples are used to test the generalization and therefore are called test data.

For example, a simple problem is to fit a line to a given set of example data points. To represent this line, the first-degree polynomial can be used, as shown in Eq. (2.1).

$$f(x) = mx + b \tag{2.1}$$

The goal is to find values for the parameters m and b that result in an accurate approximation of f with respect to the arguments x. Substituting the parameters with θ and adding subscripts results in Eq. (2.2).

$$f(x) = \theta_1 x_1 + \theta_0 \tag{2.2}$$

Further generalizing this equation yields Eq. (2.3), given $x_0 = 1$.

$$f(x) = \sum_n \theta_n x_n \tag{2.3}$$

The line fitting example is very similar to an ANN class, the multilayer perceptron (MLP). In fact, Eq. (2.3) shows the linear activation function calculation for a

neuron of a linear MLP. The subscript i in Eq. (2.3) signifies that for a neuron multiple inputs can be used. Multiple neurons for the same inputs form a layer. If all inputs are connected to all neurons this layer is called a fully-connected layer. These layers can be stacked together consecutively. For a linear MLP, multiple layers can be reduced to a layer. This implies that a linear MLP can only approximate linear functions. To approximate non-linear functions, non-linear activation functions are used.

2.1.1 Types of Training

The training of ML techniques can be categorized into different types (Russell and Norvig, 2010). For this work, the differentiation between supervised, unsupervised and semi-supervised learning is important. In supervised learning, the training provides input features and expected output labels. A supervised ML technique is optimized to learn a mapping between the features and labels. In unsupervised learning, the training provides only input features and the ML technique is optimized to find patterns within this data. Semi-supervised learning combines both, supervised and unsupervised learning (Chapelle *et al.*, 2010). It often refers to a learning task where a small amount of training data with labels is used to handle more training data without labels. In this work semi-supervised learning only refers to the combination of supervised and unsupervised learning in that unsupervised methods learn patterns that are used within a supervised method.

2.1.2 Activation Functions

An activation function ϕ maps values into the desired range. During the experimentation phase of this work sigmoid and rectified activation functions were used. For the results in this work, only the rectified activation functions are used.

The sigmoid function is a monotonic curve that is bound by an upper and a lower horizontal asymptote. Meaning it is either only increasing from the lower to the upper horizontal asymptote or vice versa. One important sigmoidal activation function is the logistic function shown in Eq. (2.4).

$$\phi(x)_{\text{sig}} = \frac{1}{1 + e^{-x}} \tag{2.4}$$

The logistic function is bound by $(0, 1)$, looks like a smooth step function, and is often used for outputs that expect a probability value.

Another important sigmoidal activation function is the hyperbolic tangent function shown in Eq. (2.5).

$$\phi(x)_{\text{tanh}} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

The hyperbolic tangent function is bound by $(-1, 1)$ and symmetric to the origin. Based on the derivative of sigmoidal functions, their gradient is generally small. In multilayer networks, the gradient decreases from layer to layer up to insignificance. This issue is called the vanishing gradient.

The rectifier function is a piecewise function that maps the identity for positive values without upper bound and rectifies negative values towards zero. This function is also a non-linear function, despite its partial linearity. Since the derivative of rectifier functions is one for positive values the gradient does not vanish. The rectified linear unit (ReLU) (Nair and Hinton, 2010) shown in Eq. (2.6) is the standard rectifier function that returns the identity for positive values and zero for negative values, therefore being bound by $[0, \infty)$.

$$\phi_{\text{ReLU}}(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.6}$$

While a ReLU has no vanishing gradient, the derivative for negative values is zero and therefore the gradient is also zero. This problem is called a dying ReLU and is solved with other rectifier variants.

One variant is the leaky ReLU (Maas *et al.*, 2013) shown in Eq. (2.7).

$$\phi_{\text{Leaky ReLU}}(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.7}$$

Contrary to a ReLU the leaky ReLU also maps negative values downscaled by a constant weight, therefore being bound by $(-\infty, \infty)$ instead.

Another variant is the exponential linear unit (ELU) (Clevert *et al.*, 2015) shown in Eq. (2.8).

$$\phi_{\text{ELU}}(x) = \begin{cases} a(e^x - 1) & \text{for } x \leq 0 \quad \text{with } a \geq 0 \\ x & \text{for } x > 0 \end{cases} \tag{2.8}$$

The ELU exponentially decreases towards a lower horizontal asymptote for high negative values, therefore being bound by $(-a, \infty)$ with $-a$ being the horizontal position of the asymptote.

Softplus (Glorot *et al.*, 2011) is a smooth variation of the ReLU and therefore similarly bound by $(0, \infty)$, with zero as an asymptote. The equation of softplus shown in Eq. (2.9) is not piecewise and therefore not directly a rectifier function.

$$\phi_{\text{Softplus}}(x) = ln(1 + e^x) \tag{2.9}$$

2.1.3 Gradient Descent Optimization

The function approximation through a parameterized model is a parameter optimization problem. Given a random parameter assignment, the function output for examples completely differs from the expected results. This difference is referred to as loss and is used to determine the partial responsibility of every parameter regarding this loss. This process is called backpropagation and the partial responsibilities as a whole are the gradient (Linnainmaa, 1970, 1976). Formally, as shown in Eq. (2.10), the gradient of the loss L with respect to the parameters θ, denoted $\nabla_\theta L$, is a vector of all partial derivatives (Shalev-Shwartz and Ben-David, 2014).

$$\nabla_\theta L = \left(\frac{\partial L}{\theta_1}, \ldots, \frac{\partial L}{\theta_n} \right) \tag{2.10}$$

The function and its parameters are hereby represented as an ANN. Gradient descent (Cauchy, 1847; Hadamard, 1908; Kelley, 1960) is an iterative optimization that tries to minimize the loss by going towards the negative gradient. The parameter update for a gradient decent step is shown in Eq. (2.11),

$$\Delta\theta = -\eta \cdot \nabla_\theta L \tag{2.11}$$

where the learning rate η regulates the parameter change. Since backpropagation for ANNs proved to work well (Werbos, 1974), it is the de facto standard for the current generation of ANNs.

For this work, it is important to mention that multiple losses for one function or parts of that function can be optimized in parallel. There are two ways to optimize such a case, alternate the optimization between these losses or optimize a joint loss.

Using batches of training examples is an established practice to reduce the time to convergence due to a smoother loss surface with a slight tradeoff in generalization (Clearwater *et al.*, 1989). It is also common to shuffle the training data to achieve a stochastic process, therefore called stochastic gradient descent (SGD) (Robbins and Monro, 1951). Adaptive moment estimation (Adam) (Kingma and Ba, 2014) is an SGD variant used in this work that smooths the gradient by estimates of its mean and uncentered variance. The parameter update for an Adam step is shown in Eq. (2.12),

$$\begin{aligned} m' &= \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta L \\ v' &= \beta_2 \cdot v + (1 - \beta_2) \cdot \nabla_\theta L^2 \\ \Delta\theta &= -\eta \cdot \frac{1}{\sqrt{v'} + \epsilon} \cdot m' \end{aligned} \tag{2.12}$$

where m' and v' are exponentially moving averages of the gradient and the squared gradient, respectively, with an exponential decay bayed on the hyperparameters β_1 and β_2.

2.1.4 Regularization

If a trained ANN is overfitting, it performs well (i.e. has a low loss) on the training data but performs worse (i.e. has a higher loss) on the test data. An overfitting network did not learn a generalized function of the objective but rather learned characteristics of the training data (Poole and Mackworth, 2017). Generalization is one important property that the learned function should satisfy. Regularization is a technique often used to support ANNs to satisfy such properties. The regularization can, for instance, be done by adjusting the loss definition or by defining a joint loss of the original objective and a regulating objective (Goodfellow *et al.*, 2017). Another type of regularization is to control neuron activity. There are many more types of regularization but they are irrelevant for this work and hence not mentioned here.

2.1.5 Normalization

Besides using regularization, normalization can adjust the data to satisfy specific properties. This section introduces some normalization functions, denoted by ψ.

Min-max normalization is a technique that adjusts the value range while preserving the proportions. The min-max normalization as shown in Eq. (2.13) scales the data into the range $[0, 1]$ (Shalev-Shwartz and Ben-David, 2014).

$$\psi_{\text{min-max}}(x) = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.13}$$

Standardization as shown in Eq. (2.14) is another normalization technique that adjusts the data to have a zero mean and a unit standard deviation (Shalev-Shwartz and Ben-David, 2014).

$$\psi_{\text{standard}}(x) = \frac{x - \mu(x)}{\sigma(x)} \tag{2.14}$$

Data in the range $[0, 1]$ can be transformed to any range with a scaling term γ and a shift term β as shown in Eq. (2.15).

$$\mathit{transform}(x) = \gamma \cdot \psi(x) + \beta \tag{2.15}$$

Batch normalization is a popular normalization method that uses standardization on batch-level and this transformation with γ and β as learned parameters (Ioffe and Szegedy, 2015).

2.1.6 Windowing

Windowing is a partition operation that splits the input data into smaller evenly shifted windows. The window size defines the size of the windows and the stride defines the size of the shifts. Depending on the stride, the windows overlap, are

side by side or have spacing between them. Overlapping means that single values can be present in multiple windows. Spacing means that the values not covered by windows are not included.

If the window size with the given stride does not fit the input data without overlapping an edge, padding is used. The most common padding variants are zero-padding an no-padding. Zero-padding adds zeros to the edge of the input data until the window size with the given stride fits. Without padding only the data area for which the window size with the given stride fits is covered, the remaining data area is ignored.

2.2 Filter Learning

In image processing, a filter can be used, for example, to sharpen or blur an image or to highlight edges. These effects can be achieved by applying a kernel to an image. Such a kernel is typically a small matrix (e.g. 3×3 px or 5×5 px) that contains a special pattern. The discrete convolution operation takes the sum of the pointwise multiplications between the kernel matrix and the reversed window matrix (le Rond d'Alembert, 1754). For a whole image, discrete convolution is applied to every window. The discrete convolution operation ($*$) for a pixel with position x and y a window of the image I and a kernel matrix K is defined in Eq. (2.16).

$$I[x, y, m] = K * I = \sum_{c} \sum_{u=-i}^{i} \sum_{v=-i}^{i} K[u, v, c] \cdot I[x - u, y - v, c] \tag{2.16}$$

The kernel matrix is hereby indexed from $-i$ to i and the respective indices u and v are applied in a way that the window is reversed. The result for a pixel is the sum of all pixelwise products and all channels c. For example, the application of a Sobel kernel (Sobel and Feldman, 1968) to highlight diagonal edges is shown in Fig. 2.1.

Instead of using handcrafted kernels like the Sobel kernel, the kernel values can be treated as parameters and therefore optimized for a given task (Fukushima, 1980). As with an MLP, backpropagation can be used to perform this optimization (LeCun *et al.*, 1989). For an image, multiple kernels can be learned to increase the expressiveness. Such a filter, consisting of multiple kernels and the convolutions thereof is called convolutional layer. As with an MLP layer, the output of a convolutional layer is adjusted by a non-linear activation function. The CNN architecture consists of multiple convolutional layers (LeCun *et al.*, 1999).

The actual implementation of a convolutional layer is often using the cross-

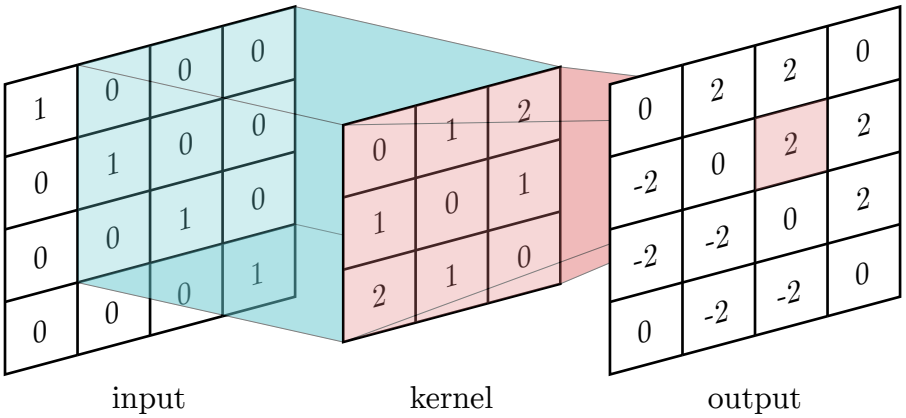

Figure 2.1: Discrete convolution of a Sobel kernel matrix and a window of size 3×3 px to highlight diagonal edges for an input image of size 4×4 px with a stride of one.

correlation operation as shown in Eq. (2.17) (Goodfellow *et al.*, 2017).

$$I[x, y, m] = K * I = \sum_{c} \sum_{u=-i}^{i} \sum_{v=-i}^{i} K[u, v, c] \cdot I[x + u, y + v, c] \tag{2.17}$$

The only difference between discrete convolution and cross-correlation is that in cross-correlation the window is not reversed. Since the kernel parameters are learned, reversing the window is unnecessary and just produces overhead. This work will refer to cross-correlation as convolution.

2.3 Pooling

In CNNs, pooling is an operation to reduce the dimensionality and the effect of shifts and distortions (Goodfellow *et al.*, 2017). Dimensionality reduction can be necessary to ensure computational efficiency when high dimensional data is used. Reducing the effect of shifts and distortion increases the robustness.

Pooling is a non-linear operation that is applied on a per-window basis, again using the windowing operation. It reduces all input values from one window to a scalar value. Unlike in convolution, pooling is applied on a per-channel basis. Compared to a convolutional layer, pooling is a fixed method without any learnable parameters. Taking the mean, minimum, or maximum are the most commonly used reduction methods. Depending on the reduction method, the operations are called max-pooling (Zhou and Chellappa, 1988), min-pooling, or average-pooling. For example, the max-pooling function shown in Eq. (2.18), reduces a window to its max value per channel.

$$I[x, y, c] = max(w) \quad \text{where } w = \begin{pmatrix} I[x-i, y-i] & \cdots & I[x+i, y-i] \\ \vdots & \ddots & \vdots \\ I[x-i, y+i] & \cdots & I[x+i, y+i] \end{pmatrix} \tag{2.18}$$

2.4 Representation Learning

For humans, it is way easier to solve a problem if it is represented in a way that can be directly approached. For example, every human tasked to read a digital document would struggle if it is represented by bits instead of a character encoding. Similarly, ANNs learn intermediate representations that support its structure to solve a given task.

This representation could be sparse, i.e. have a high ratio of zero values to non-zero values. Such sparsity in an internal representation stems from either a sparse activity or sparse connectivity of neurons (Thom and Palm, 2013).

Variations in a data distribution can be disentangled into representations of explanatory factors (Bengio *et al.*, 2013). In contrast to sparsity, the disentanglement property directly refers to the information contained within the representation. A representation, therefore, can be disentangled without being sparse. Thus far, there is no consensus on what defines a disentangled representation (Higgins *et al.*, 2018; Mathieu *et al.*, 2019). This work uses a restrictive definition in which a fully disentangled representation is defined as a representation that captures at most one explanatory factor per value (Eastwood and Williams, 2018; Kim and Mnih, 2018).

This work exclusively addresses the representations learned by autoencoders (AEs) and therefore introduces AEs and multiple used variants in this section.

2.4.1 Standard Autoencoder

Encoding is a process that transforms data into another representation. In AEs the encoding process targets a reduced representation that preserves important properties (Goodfellow *et al.*, 2017). Since it is not clear what the important properties are, AEs use a trick. Instead of directly encoding some input, the goal is to reconstruct the input. With a bottleneck at the center, the AE learns a code representation that preserves all properties that are necessary to reconstruct the input and throws away the rest. The important properties are not predefined and for that reason, this is an unsupervised training method.

The success of a reconstruction is defined by the similarity compared to the input data. The similarity definition, therefore, determines which properties are important for a successful reconstruction. For humans, these properties are the explanatory factors of the input data. The properties learned by a standard AE

can be inexplicable but have to contain the information that most defines a given sample with respect to the training data.

This work exclusively uses pixel distance-based similarity metrics. These metrics are the mean absolute error (MAE), the mean squared error (MSE), and the log loss (Goodfellow *et al.*, 2017; Shalev-Shwartz and Ben-David, 2014). Since in the context of ANNs the difference between input x and reconstruction y is called loss L, the respective metric is called loss function $L(x, y)$. The loss function is evaluated for every pixel i and averaged over all N pixels.

The MAE as shown in Eq. (2.19) calculates the average distance between two data samples.

$$L(x, y) = \frac{1}{N} \sum_{i=1}^{N} |y_i - x_i| \tag{2.19}$$

In addition to MAE, the MSE as shown in Eq. (2.20) especially highlights larger distances.

$$L(x, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i)^2 \tag{2.20}$$

The log loss as shown in Eq. (2.21) is based on probability theory and therefore only works for input and reconstruction pixel values in the range of $[0, 1]$.

$$L(x, y) = \frac{1}{N} \sum_{i=1}^{N} -(x_i \log(y_i) + (1 - x_i) \log(1 - y_i)) \tag{2.21}$$

This loss especially punishes if the model makes wrong predictions it is certain about and pushes predictions towards binary decisions.

2.4.2 Encoder and Decoder Architecture

The encoder performs a data reduction and the decoder performs a data expansion. Methods used within the encoder and decoder need to support the respective size change. Since in a fully-connected layer, as introduced in Section 2.1, the number of used neurons defines the output size, it can be used for the encoding and decoding process. The output size of a convolutional layer, as introduced in Section 2.2, is smaller or equal to the input size depending on the stride size. The convolutional layer, therefore, can only be used for the encoding process. The decoder counterpart to the convolutional layer is the transposed convolutional layer. The result of the transposed convolution is the gradient of a convolutional layer with the same hyperparameters (Dumoulin and Visin, 2016). Common AE implementations use some combination of fully-connected and convolutional layers for the encoder as well as some combination of fully-connected and transposed convolutional layers for the decoder.

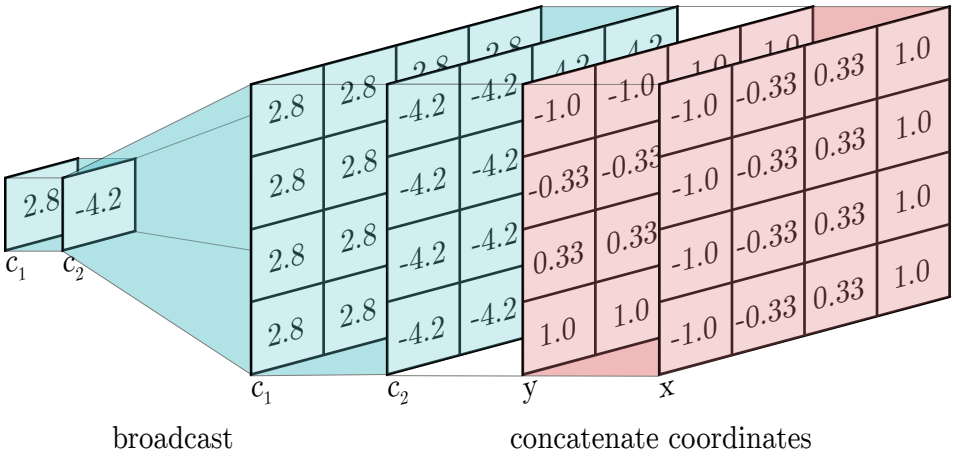

Figure 2.2: Spatial Broadcast for an input vector c and a target size of 4×4 px concatenated with the x and y coordinates of range $[-1, 1]$.

In addition to the common encoder-decoder variants, this work uses the Spatial Broadcast decoder (Watters *et al.*, 2019). At its core, the Spatial Broadcast decoder is an alternative to the transposed convolution. Instead of learning how to distribute the code to render objects at correct spatial positions, the codes are broadcasted and supplemented by their spatial position. That reduces the decoding task to a kind of threshold function for objects at every spatial position. More precisely, the Spatial Broadcast method takes a vector c as input and tiles it to a target output size. Tiling hereby refers to duplicating c in the x and y dimension until it has the correct width and height. This tensor is extended by concatenating the x and y coordinates of every point in the channel dimension. The coordinates are linearly distributed over the range $[-1, 1]$. An example application of this method is shown in Fig. 2.2.

In this work, fully-connected layers, convolutional layers, and the Spatial Broadcast decoder are abbreviated in graphics and plots with *fc*, *conv*, and *sb*, respectively.

2.4.3 Activation Restricted Autoencoder

The goal of a k-sparse AE (Makhzani and Frey, 2013) is to produce sparse code representations. To achieve sparsity the k-sparse AE simply reduces the code representation to its k largest values. Therefore the top-k selection function $\text{top}_k(z)$ that returns the indices of the k largest activations of the code representation z is introduced. During training, only the k largest activations are used and the rest is set to zero. During testing Makhzani and Frey (2013) observed that $\text{top}_{ak}(z)$ results in better results for subsequent classification objectives, where $a \geq 1$ is manually selected. The application of this process is formulated in Eq. (2.22).

$$z_i = \begin{cases} z_i & \text{for } i \in \text{top}_k(z) \quad \text{when training} \\ z_i & \text{for } i \in \text{top}_{ak}(z) \quad \text{when testing} \\ 0 & \text{else} \end{cases} \qquad \text{where } a \geq 1 \tag{2.22}$$

2.4.4 Variational Autoencoder

The objective of a variational autoencoder (VAE) (Kingma and Welling, 2013) is to find an isotropic multivariate Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$ for the code representation z from which samples like the input are likely. Since a Gaussian distribution can be described by its mean μ and standard deviation σ i.e. $\mathcal{N}(\mu, \sigma^2)$, the encoder is trained to learn these two parameters. Given such a distribution that captures the essentials of the input data, the parameters μ and σ of the distribution can be used to draw new samples from this distribution. As shown in Eq. (2.23), the sampling method uses a random value ϵ drawn from an isotropic multivariate standard Gaussian $\mathcal{N}(0, I)$ to generate new samples from μ and σ.

$$z = \mu + \sigma \odot \epsilon \quad \text{where } \epsilon \, \mathcal{N}(0, I) \tag{2.23}$$

As regularization, the Kullback–Leibler (KL) divergence is used to reduce the divergence between the distribution generated for z and an isotropic multivariate standard Gaussian $\mathcal{N}(0, I)$ as shown in Eq. (2.24).

$$KL(\mu, \sigma) = \sum_{j=1}^{J} \mu_j^2 + \sigma_j^2 - \log(\sigma_j^2) - 1 \quad \text{where } J = \#\text{code variables} \tag{2.24}$$

Minimizing this divergence restricts the code representation capacity, with the representation z not holding any information about the input x when the divergence is zero (Burgess *et al.*, 2018). This regularization can be seen as an information bottleneck and as a consequence of fewer features encoded in the representation as a way to improve statistical independence between code variables (Higgins *et al.*, 2017; Burgess *et al.*, 2018). Intuitively the goal of this independence is that every code variable captures another independent feature of the input data i.e. the disentanglement of the code representation. The total loss function $L(x, y)$ combines the KL divergence regularization $KL(\mu, \sigma)$ and a reconstruction loss $RL(x, y)$ as shown in Eq. (2.25).

$$L(x, y) = \frac{1}{2} KL(\mu, \sigma) + RL(x, y) \tag{2.25}$$

2.4.5 Capacity-restricted Variational Autoencoder

Since the KL divergence term is responsible for the information bottleneck, increasing its weight leads to fewer features being encoded in the code representation (Burgess *et al.*, 2018). The fewer features encoded in the representation the more statistically independent the code variables appear (Higgins *et al.*, 2017). To increase the weight of the KL divergence term the β term is added to the standard VAE formulation (Higgins *et al.*, 2017). As shown in Eq. (2.26), β is a simple scalar weighting factor of the KL divergence term (Higgins *et al.*, 2017).

$$L(x, y) = \beta KL(\mu, \sigma) + RL(x, y) \quad \text{where } \beta > 1 \tag{2.26}$$

For higher values of β, a better disentanglement of the code representation can be perceived (Higgins *et al.*, 2017). The tradeoff of this gain in perceived disentanglement is a loss in reconstruction quality such as blurrier reconstructions and the sacrifice of minor features in the code representation (Higgins *et al.*, 2017).

2.4.6 Capacity-controlled Variational Autoencoder

Given that a low KL divergence to an isotropic multivariate standard Gaussian restricts the code representation capacity, then a high divergence allows a high capacity (Burgess *et al.*, 2018). Hence, gradually increasing the divergence also allows the code representation to gradually encode more features (Burgess *et al.*, 2018). To achieve this effect, the KL divergence is controlled by a term C which is gradually increased during training (Burgess *et al.*, 2018). This new loss function is shown in Eq. (2.27),

$$L(x, y) = \gamma |KL(\mu, \sigma) - C| + RL(x, y) \tag{2.27}$$

where γ ensures that the actual divergence is close to the control value C (Burgess *et al.*, 2018).

2.5 Spatial Transformations

Images are a projection of the shape and visual appearance of objects within a scene. For humans, it is relatively easy to identify an individual object independent of its size, rotation, position, and lighting (Shepard and Metzler, 1971). ML algorithms optimally should be able to detect individual objects independent of these types of spatial transformations. This work differentiates between four spatial transformations. These transformations are the translation, rotation, scale, and illumination.

Translation, rotation, scale, and illumination are transformations that refer to a position, angle, size, or lighting change of an object within an image, respectively. An ML algorithm should detect an object in an image independent of its position or location of the camera, its orientation or the orientation of the camera, its size or distance to the camera, and its color properties, lighting or the general brightness (Nixon and Aguado, 2013).

2.6 Representational Properties

Representation learning can be seen as a function ϕ that maps input data x to a representation z, as shown in Eq. (2.28).

$$z = \phi(x) \tag{2.28}$$

Equivariance and invariance are two representational properties that state the effect of spatial transformations t within x on z (Lenc and Vedaldi, 2014).

The representation mapping process is invariant to spatial transformations if these do not affect the code representation (Lenc and Vedaldi, 2014). Formally, if the mapping $\phi(t(x))$ has the same result independent of the transformation T, then ϕ is invariant to T, as shown in Eq. (2.29) (Shen, 2017).

$$\phi(T(x)) = \phi(x) \tag{2.29}$$

For common ML tasks like classification, spatial transformations are irrelevant and the ability to ignore them is desirable (Shen, 2017). For example, for the task of detecting cats in images, it is only relevant if a cat is present and not for instance where the cat is.

The representation mapping process is equivariant to spatial transformations if the transformation is encoded in the representation (Lenc and Vedaldi, 2014). For the formulation of equivariance, a transformation function T' is introduced that captures how the transformation T would affect representation z (Lenc and Vedaldi, 2014). As shown in Eq. (2.30), the mapping ϕ is equivariant to the transformation T if a transformation function T' exists and applying it on z has the same result as applying the transformation T directly on x (Shen, 2017).

$$\phi(T(x)) = T'(\phi(x)) \tag{2.30}$$

For example, if it is known how the position of the cat in an image affects the representation, then the position can be read from the representation.

If a function is invariant to a transformation the transformation information is lost. If a function is equivariant to a transformation the transformation information is present in the output. An invariant representation can therefore always be

obtained from an equivariant representation but not vice versa (Kuzminykh *et al.*, 2018).

2.7 Adversarial Attacks

Since adversarial attacks are not the main focus of this work, they are only explained briefly to introduce important terms. Adversarial attacks are a technique to find input images that trick a classifier but would not trick a human. These misclassified input images are called adversarial examples (Szegedy *et al.*, 2013). Finding adversarial examples is done by applying small perturbations to correct classified input images until they are misclassified. The goal of such an adversarial attack can either be targeted at tricking the classifier into classifying a specific class or untargeted, tricking the classifier into classifying any class except the correct one (Yuan *et al.*, 2017). These attacks need limitations to produce perturbations that only affect the classifier but not a human (Yuan *et al.*, 2017). In other words, the perturbations should not change the input image so much so that it represents a different class.

Adversarial examples highlight problematic decision boundaries (Moosavi-Dezfooli *et al.*, 2016), i.e. instabilities. Adversarial attacks can, therefore, be used to test the robustness of ANN.

Chapter 3

Locally Embedded Autoencoder

While the hierarchical structure of discriminative objects first and foremost refers to the semantic difference between those objects, they can also be differentiated by their size. For subobjects to be within an object, they naturally have to be smaller than their parent object. To achieve this local view of subobjects, AEs are applied to local windows extracted by a windowing operation. Based on this spatial locality, the operation presented in this work is referred to as locally embedded autoencoder (LEA).

Since this work focuses on image data, the input is represented as a tensor $[w, h, c]$ where w is the width, h the height and c the channel dimension. However, the described operation could be used on data of any dimensionality.

In Eq. (3.1), an AE is defined as a function ae for all values of an input window, including all channels.

$$I[x, y, 0 \dots C] = ae(w) \quad \text{where } w = \begin{pmatrix} I[x-i, y-i] & \cdots & I[x+i, y-i] \\ \vdots & \ddots & \vdots \\ I[x-i, y+i] & \cdots & I[x+i, y+i] \end{pmatrix} \tag{3.1}$$

In this formula, C refers to the code length within the AE. In a LEA every window is encoded by one AE.

The convolutional layer, max-pooling layer, and the LEA, applied to a whole image, have the same dimension reduction effect for the width and height dimension depending on stride and padding. Their behavior for the channel dimension differs. Standard max-pooling with no stride on the channel dimension has the same number of output channels as input channels. Convolution reduces all input channels to one channel per kernel, resulting in the same number of output channels as used kernels. The AE maps all channels to one code, with the code length being a hyperparameter. The convolutional layer and the LEA are similar in that they both allow the definition of the number of output channels, i.e. by defining the number of kernels to use or by defining the code length.

The encoding process of windows via a LEA with the resulting output structure is shown in Fig. 3.1. The output of a convolutional layer has a similar structure with every channel representing the output for a kernel. The output image of a

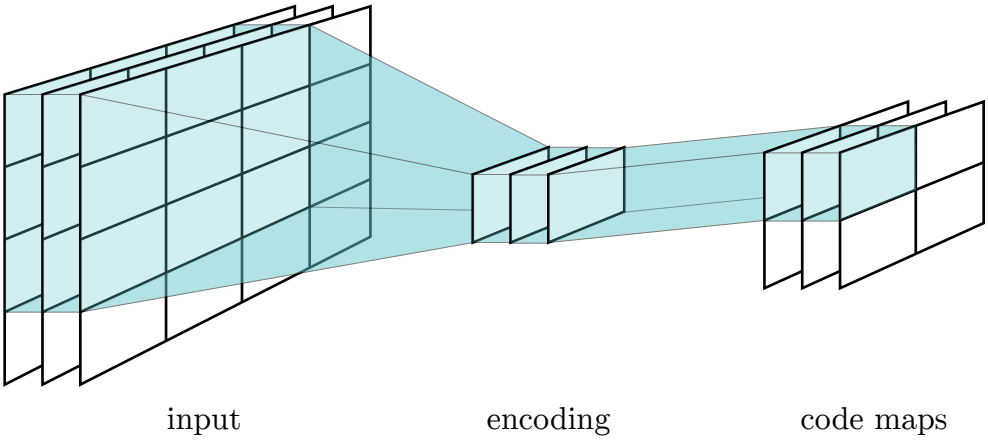

Figure 3.1: Encoding of windows via a LEA. Recombining all codes results in code maps (in this example no-padding is applied).

kernel is called a feature map and visualizes the effect of this kernel on the whole image. Therefore, combining the output of all kernels results in a set of all feature maps. The channel dimension of the LEA output represents the results of one code variable. The output image for a code variable is called a code map and visualizes the information encoded by this code variable. All codes combined result in a set of code maps. In other words, the LEA can be viewed as an unsupervised convolutional layer.

Locally embedded instead of globally embedded AEs are used for two reasons — computational efficiency and spatial locality.

The input data size has a substantial influence on the number of parameters in a directly applied AE. With more free parameters the training time increases significantly. Using the LEA restricts the number of parameters to the window size. While windowing results in a set of windows and an AE is applied to every window, only one AE is used. This makes the AE position-independent. Furthermore, this vastly reduces the number of parameters and therefore the training time. Additionally, instead of training one AE on all windows in parallel with shared weights, the windows are used in a batch fashion. The expected training input for a LEA is of the dimensions $[b, w_i, h_i, c]$ with batch size b, image width w_i, image height h_i and the number of channels c. The training input for the single AE after the windowing process is of dimension $[b \cdot W, w_w, h_w, c]$ where W is the number of windows, w_w the window width and h_w the window height. This results in an additional efficiency improvement since the parallel training for every window only needs a synchronization on the loss calculation and not on the backpropagation.

3.1 Integrability

As shown in Fig. 3.2, it is possible to integrate a LEA into any ANN architecture. This is due to considerations to multi-task learning and unrestricted inputs.

3.1.1 Multi-task Learning

The process of training an ANN that has LEAs can be thought of as multi-task learning (MTL). The main task has an objective, for example getting good at classification. Every LEA has the objective of learning a local disentangled representation and is considered as an additional task.

The optimization of multiple tasks can be done jointly or alternately. The joint optimization of the two tasks is useful if both tasks are related (Baxter, 1995). In a joint optimization, all parameters are adjusted based on both objectives or a combined objective. It is important to point out that it can be difficult to balance the weighting of the loss contributions in proportion to the combined loss. One loss in a combined loss can dominate the other loss, such that the corresponding task of the dominated loss is not learned effectively. In an alternate optimization, two sets of parameters are adjusted based on their objectives. An alternate optimization of dissimilar subtasks might be necessary if a joint optimization leads to a negative transfer (Kang *et al.*, 2011).

Solving the main task requires the LEA to at least preserve information relevant to the main task. From the perspective of the optimization of the main task, the representation created by the LEA is optimal if it is optimized specifically for the main task. From this it follows that the LEA objective is neglected, the decoder is ignored, and the encoder just adds additional free parameters to the model for solving the main task. This does not mean that both objectives contradict, it only illustrates that the LEA objective acts as a regularization technique for the main task instead of a fully common objective. This property points out that a joint optimization with a combined objective could be problematic. Firstly, while both objectives might not contradict each other, they still can destabilize the training process. The combined loss of the main task and the LEAs could have a rough surface, therefore being hard to optimize. Secondly, the main task could dominate the LEA objective, thereby violating the intention of this work. Additional weighting hyperparameters and an elaborate tuning of these would be required. Both issues make the evaluation of this operation difficult, could cause side effects and therefore lead to wrong conclusions. Based on the effect of the main task on the LEA objective, an alternate training for both objectives is used.

The objective of a LEA is specifically to encode local information. Allowing the LEA optimization to adjust previous main task layers or the encoder of previous LEA would break this local view. It is not clear how the local encoding objective would effect previous layers. This could have a stabilization effect on the input

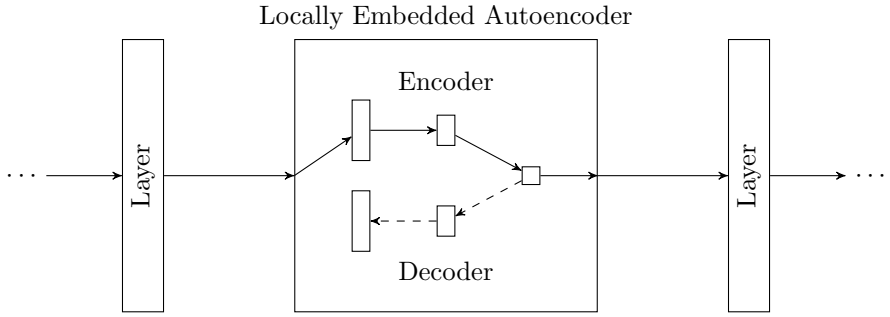

Figure 3.2: A LEA between two artificial neural network layers

values of the LEA. Most definitely, it again introduces side effects to the training of both tasks This is an additional reason for the use of an alternate training, also between multiple LEAs.

The implementation of this work supports all training variants. In Fig. 3.2, a LEA with encoder and decoder and the previous and subsequent layer of the main task model are shown. The parameters of the entire model are split into multiple subsets. Each LEA is a separate subset of parameters. All remaining parameters of the entire model not in any LEA parameter subset is part of the main task parameter subset. The loss of the main task is a scalar loss. In this work, the main task is a classification objective and the loss is the log loss between the labels and the softmax-normalized output of the main model. Every single LEA has a separate loss defined by the AE. For all AE types used in this work, reconstruction loss is averaged. To be exact, the mean of the sum of squared differences between input window w_i and reconstruction r_i of all windows $b \cdot W$ as shown in Eq. (3.2),

$$RL(w, r) = \frac{1}{l} \sum_{i=1}^{l} \sum_{u=1}^{\#w} (r_{i,u} - w_{i,u})^2 \quad \text{where } l = b \cdot W \tag{3.2}$$

where b is the batch size and W the number of windows per input defines the total reconstruction loss $RL(w, r)$ of a LEA. The summation of squared differences is done for the number of values within a window $\#w$.

For every parameter subset, there is a respective loss. The joint optimization with a combined objective is the optimization of all parameters with a weighted sum of all losses. The joint optimization with independent objectives is an optimization of all parameters with independent optimization steps per loss. The alternate optimization is an independent optimization of every parameter subset with the respective loss. It is important to note that the optimization order for the alternate optimization matters. While the optimization of LEA parameters is fully independent of main model parameters during an optimization step, the optimization of the main model parameters directly depends on the LEA parameters. This

is due to the error of the main model parameters being backpropagated through the encoder of the LEA. The parameter subsets, therefore, need to be updated sequentially. Firstly, the main model parameters are updated. Secondly, the parameter subsets of every LEA are updated in parallel.

To reiterate, the optimization of the main task and the LEA objective is an MTL process. Based on their common objective, it makes sense to optimize them jointly. Still, to avoid side effects and for an accurate evaluation of the LEA, they are mainly optimized alternately.

3.1.2 Unrestricted Inputs

If an AE is directly applied to samples, the input value range is known and fixed. However, for a LEA with a preceding ANN layer, this is not the case. During training, the input value range is constantly shifting and the distribution is changing. For a LEA this entails the issue that the reconstruction value range is also unknown and the issue of instability based on the constant adjustments. Since dataset values are often normalized to a $[0, 1]$ value range, for a simple AE the reconstruction values are also expected to be within this range. This allows the usage of certain techniques, like a logistic activation function for the reconstruction output layer or the log loss as reconstruction loss. Having an unrestricted value range, these techniques are not applicable. A direct approach to use the same or similar techniques would be to restrict layers within the network. The restriction to logistic activations would enable the usage of a log loss for the reconstruction loss. The restriction to other activation functions would at least make the value ranges bounded. On the upside, restricting the activations would make the analysis easier and the training more stable. On the downside, a restriction on the activations is also a restriction on the expressiveness of the LEA. Therefore, no specific restrictions on the layers of the main model are assumed. As a consequence, LEA input values and reconstruction values are assumed to be unrestricted. To enable such unrestricted reconstructions, the last decoding layer of all AE types used in this work is a fully-connected layer with linear activation. As already specified in Eq. (3.2), the reconstruction loss is the mean of the sum of squared differences between input and reconstruction of all windows.

Another solution would be to use batch normalization on the LEA input but this would also introduce additional side effects. Improvements that come seemingly due to the LEA could be based on the inclusion of batch normalization. But, this would make the independent analysis of the effects of this operation difficult. Therefore, batch normalization is not used in this work.

3.2 Autoencoder Interchangeability

Special attention was given to the modularity regarding the AE types used within the LEA. The LEA is designed in a way that any AE type with predefined encoder and decoder types can be used. This makes a flexible experimentation and analysis phase possible. For a LEA, the AE model is provided as a parameter. Therefore, the LEA implementation can be divided into two parts, an AE and a wrapper.

The wrapper prepares the input data for the AE and post-processes the codes. In the preparation step, the layer input is split into windows which are flattened to the batch level. In the post-processing step, batch samples and per-window codes are reshaped to a batch of code maps.

The input of the AE is, therefore, a batch of windows. The only requirement for the AE implementation is again that input values are unrestricted and need to be handled accordingly. In this work, two encoder- and three decoder-types are provided which can be set and combined via parameters. For both, the encoder and the decoder, there exists a fully-connected and a convolutional variant. As a decoder, additionally, the Spatial Broadcast variant is provided.

On a further technical note, only the code maps are returned for a LEA. This is an implementation detail that allows to build up a network graph without handling the AE loss. Instead, all AE losses are added to a global collection. Only at the definition of the optimization, these losses are retrieved.

3.3 Autoencoder Types

For the LEA, the AE type is considered as elementary to obtain good codes. The selection is, therefore, an important part of this work. Two properties are important for this selection, the encoding and disentanglement capabilities. For this work, four different AE types were considered. These are the standard AE, the k-sparse AE, the VAE, and the capacity-controlled VAE. From the two encoder types and three decoder types, a selection of three combinations are evaluated. These three are a combination of the fully-connected variants, a combination of a convolutional encoder with a transposed convolutional decoder, and a combination of a convolutional encoder with a Spatial Broadcast decoder. In this work, these are abbreviated as *fc*, *conv*, and *sb*, respectively.

3.3.1 Encoder-decoder Combinations

The encoder and decoder layers are implemented generically. The layers automatically adapt to the given input and do not need hyperparameter tuning regarding the input size. In this work, all layers use a ReLU activation function and bias neurons.

The fully-connected encoder has no additional hyperparameters. The encoder takes the input window as a flat vector and processes it with a fully-connected layer two a vector twice as big. The actual encoding is part of the AE specific part and the encoder layer is a pre-encoding step. This is true for all encoder types.

The fully-connected decoder is similar to the reverse of the fully-connected encoder and also has no additional hyperparameters. The decoder takes the code and again processes it with a fully-connected layer to a vector twice as big. As the final decoding layer, a fully-connected layer with a linear activation function maps onto a vector of the original window size. In all decoders, the final linear activation function is necessary to shift the reconstruction values into the unrestricted value range of the input window.

The convolutional encoder has a hyperparameter, the number of kernels K. The usual convolution hyperparameters, kernel size, stride, and padding, are optional and default to a kernel size of four, a stride of two, and the zero-padding. All convolutional layers within this encoder use the same hyperparameters. The goal of the convolutional layers is to reduce the input window size w in terms of width and height to $f < k$ where f is the output size and k is the kernel size. The number of convolutional layers n needed for this reduction is calculated by Eq. (3.3).

$$n = \begin{cases} 0 & \text{for } k > w \\ \left\lceil \log_s \left(\frac{w}{k}\right) \right\rceil + 1 & \text{else} \end{cases} \tag{3.3}$$

The final size f in terms of width and height is calculated by Eq. (3.4).

$$f = \begin{cases} w & \text{for } k > w \\ \left\lceil \frac{w}{s^n} \right\rceil & \text{else} \end{cases} \tag{3.4}$$

The output of the convolutional layers is processed as a flat vector with a fully-connected layer to a vector of size $K \cdot f \cdot f$. If the kernel size is larger than the supplied window, then no size reduction is performed and the convolutional encoder reduces to a sort of a fully-connected encoder.

The transposed convolutional decoder is similar to the reverse of the convolutional encoder and also has the number of kernels as hyperparameter and the same fixed kernel size, stride, and padding. The decoder takes the code and processes it with a fully-connected layer again to a vector of size $K \cdot f \cdot f$. The size in terms of width and height is then increased to a similar size as the original input window. This size increase is performed by n transposed convolutional layers, where n is again calculated by Eq. (3.3). If $s_w \notin S$ where s_w is the original input window size and $S = 2^n | n \in \mathbb{N}^*$, then the resulting size of the increase will not match the window size. Therefore, an additional fully-connected layer with a linear activation function maps onto the correct window size. If the kernel size is larger than the supplied window,

then no size increase is performed and the transposed convolutional decoder reduces to a kind of a fully-connected decoder.

The Spatial Broadcast decoder also has the number of kernels and an optional kernel size that defaults to four as hyperparameters. The first step in the Spatial Broadcast decoder is to tile the code to the original window input size as well as adding the x and y coordinates, resulting in a size of $[w_w, h_w, C+2]$, where w_w is the window width, h_w is the window height and C is the code length. On these tiles, a fixed set of two convolutional layers with a stride of one is used. To get the same number of output channels as the input window has, an additional convolutional layer with linear activation function, also a stride of one and as many kernels as original input channels are used.

3.3.2 Core Encoding Logic

Besides a few AE type-specific hyperparameters, they all have the code length as a hyperparameter. The actual core encoding logic is equal to the standard implementation of the respective AE types. To keep the implementation clean and the analysis undistorted, no additional techniques are used. A major difference to the standard implementations is the unrestricted reconstruction and the unrestricted encoding.

The standard AE adds a fully-connected layer with linear activation function that maps the encoder output onto the code length. The loss is only the reconstruction loss as defined in Eq. (3.2).

In addition to the standard AE, the k-Sparse AE performs the top-k selection on the code. The k-Sparse AE has the additional hyperparameters k and a as described in Section 2.4.3.

The VAE adds a fully-connected layer with linear activation function that is supposed to learn the mean and logarithmic standard deviation of the code's isotropic multivariate Gaussian distribution. The VAE provides the optional hyperparameters beta and warm-up, which in experiments did not prove any significant advantage and are therefore not used in the analysis part of this work. The loss is the standard VAE objective as defined in Eq. (2.25), with the reconstruction loss as defined in Eq. (3.2).

Instead of the standard VAE loss, the loss of the capacity-controlled VAE is equal to the definition in Eq. (2.27). The loss definition of the capacity-controlled VAE includes the additional hyperparameters γ and C for the weight and control value. The capacity control is only applied during training, during the testing phase the standard KL divergence regularization as defined in Eq. (2.24) is used. During training, the control value C is linearly increased from zero to the defined maximal control value. This increase is either done over the whole training period or as warm-up until an earlier epoch defined via a hyperparameter.

Table 3.1: Comparison of the number of parameters of different AEs, different encoder-decoder combinations and different input sizes. The parameter reduction rate base on the smaller input size is shown. The standard AE and VAE show comparable results to the k-sparse AE and capacity-controlled VAE, respectively. The target code length is 10 and the number of kernels is 32.

Autoencoder type	Combination	Parameters for 28×28×1 px	Parameters for 4×4×32 px	Parameter reduction
Standard VAE	*sb*	52 053	42 196	−18.94 %
Standard AE	*sb*	51 723	41 866	−19.06 %
Standard VAE	*fc*	2 509 604	1 081 876	−56.89 %
Standard AE	*fc*	2 493 914	1 071 626	−57.03 %
Standard VAE	*conv*	25 776 516	301 716	−98.83 %
Standard AE	*conv*	25 776 186	301 386	−98.83 %

3.3.3 Comparison and Test

In Table 3.1, these different variants are compared in terms of their parameter count. Since the k-sparse AE has the same number of parameters as the standard AE and the capacity-controlled VAE has the same number as the VAE, they are omitted from the comparison. As can be seen in the comparison, the AE types only make up a small difference to the number of parameters. The parameter count of all combinations primarily depends on the decoder variant. The number of transposed convolutional layers and the number of neurons in the fully-connected decoder is proportional to the input size. As a consequence, the number of parameters increases proportionally as well. Since the Spatial Broadcast decoder has a fixed number of parameters, the input size only affects the convolutional encoder of the *sb* combination. In conclusion, the Spatial Broadcast variant has the fewest parameter.

The number of parameters in this comparison does not directly correlate with the actual training time. Training time also depends on the techniques used within these variants. This is especially apparent for the Spatial Broadcast decoder. The three convolutional layers of the Spatial Broadcast decoder all work on the whole input image size with unit stride. The comparison between parameter count and training time can be seen in Table 3.2. While the Spatial Broadcast decoder has the lowest parameter count, the training time per parameter is much larger compared to the other decoder types. For the stated configurations, these decoder types take equally long to train.

Since the difference in parameter count between the AE types is minimal, they are fairly comparable per encoder-decoder combination in terms of their per epoch performance. The encoder-decoder combinations are comparable in terms of their training time but their different parameter count suggests a greater capacity. This

Table 3.2: Comparison between the number of parameters and the median time of training 800 epochs with different AEs and different encoder-decoder combinations. The standard AE and VAE show comparable results to the k-sparse AE and capacity-controlled VAE, respectively. The target code length is 10, the number of kernels is 32, and the input image size is 28×28×1 px. The system used for the time measurements is described in Section 4.9.

Autoencoder type	Combination	Number of parameters	Training time	Time per parameter
Standard VAE	*conv*	25 776 516	2 h 31 min	0.35 ms
Standard AE	*conv*	25 776 186	2 h 33 min	0.36 ms
Standard AE	*fc*	2 493 914	2 h 05 min	3.00 ms
Standard VAE	*fc*	2 509 604	2 h 06 min	3.01 ms
Standard AE	*sb*	51 723	1 h 52 min	129.92 ms
Standard VAE	*sb*	52 053	1 h 54 min	131.40 ms

could lead to better encoding and decoding capabilities or to overfitting. Neither the encoder, the decoder, nor the AE was fine-tuned in terms of architecture or optimization hyperparameters. No enhancement with proven techniques is done. A comparison therefore only reflects the implementation as used in this work without any comparability to the state-of-the-art. Hyperparameters and other configurations not mentioned for figures are listed in Appendix A.1.

The AE implementations of this work are central to the LEA analysis. To ensure that the implementations are correct, they are tested directly as simple AEs on the Fashion-MNIST dataset, i.e. without any windowing. For a sanity check, the AE implementations are tested on their reconstruction capability. As can be seen in Fig. 3.3, all implementations successfully reconstruct the given images of the Fashion-MNIST dataset. In general, the standard AE based implementations produce more accurate reconstructions but introduce artifacts. In some cases, the k-sparse AE has issues to produce good reconstructions, as can be seen in Fig. 3.3 for the pullover and top reconstructions of the k-sparse AE with convolutional encoder and decoder. The VAE based implementations generally produce more blurry reconstructions. In some cases, the VAE loses fine details as can be seen in Fig. 3.3 for the ankle boot and bag reconstructions.

For a comparison of the reconstruction performance, all four AE types with fully-connected encoder and decoder are compared in Fig. 3.4 based on their reconstruction loss. While the AE loss includes different calculations and factors per AE type, the reconstruction loss calculation is the same for all types. It only captures the mean of the squared difference between inputs and reconstructions. As already shown in Fig. 3.3, the reconstruction loss confirms that the AE implementations are correct. The k-sparse AE performs worse than the similar standard

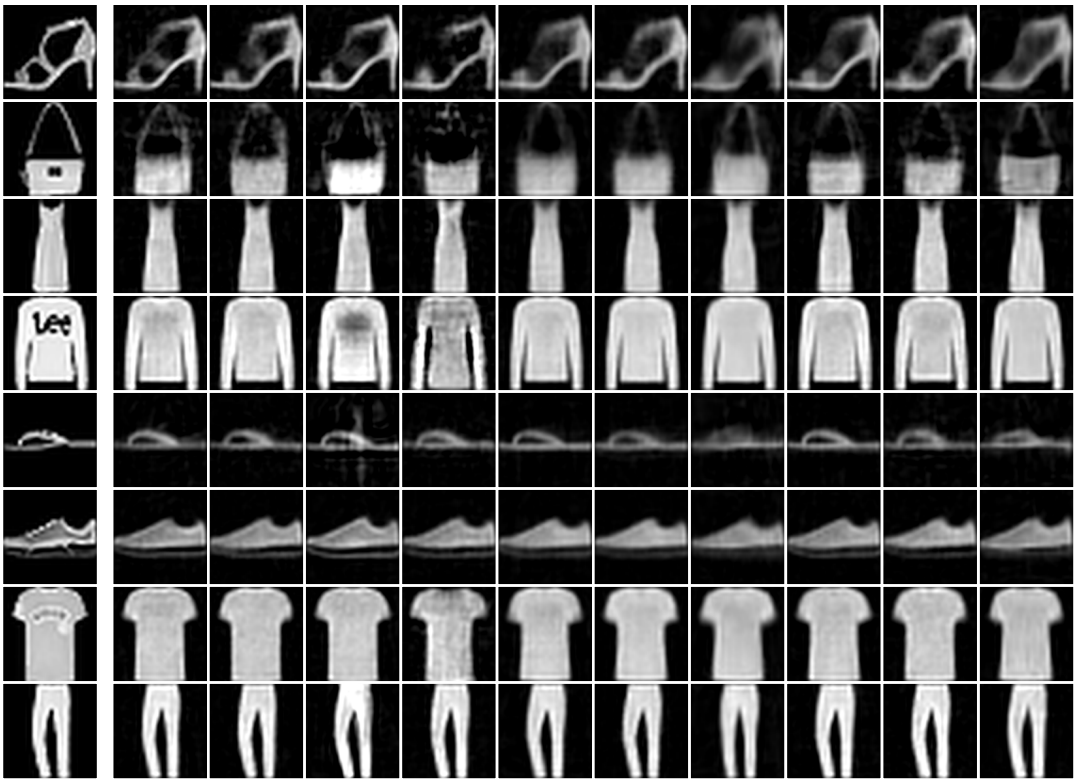

Figure 3.3: Fashion-MNIST reconstruction images with (a) being the ground truth image. The other images show the reconstructions of the following AEs : (b) Standard AE - *fc*, (c) Standard AE - *conv*, (d) k-Sparse AE - *fc*, (e) k-Sparse AE - *conv*, (f) VAE - *fc*, (g) VAE - *conv*, (h) VAE - *sb*, (i) Capacity VAE - *fc*, (j) Capacity VAE - *conv*, (k) Capacity VAE - *sb*

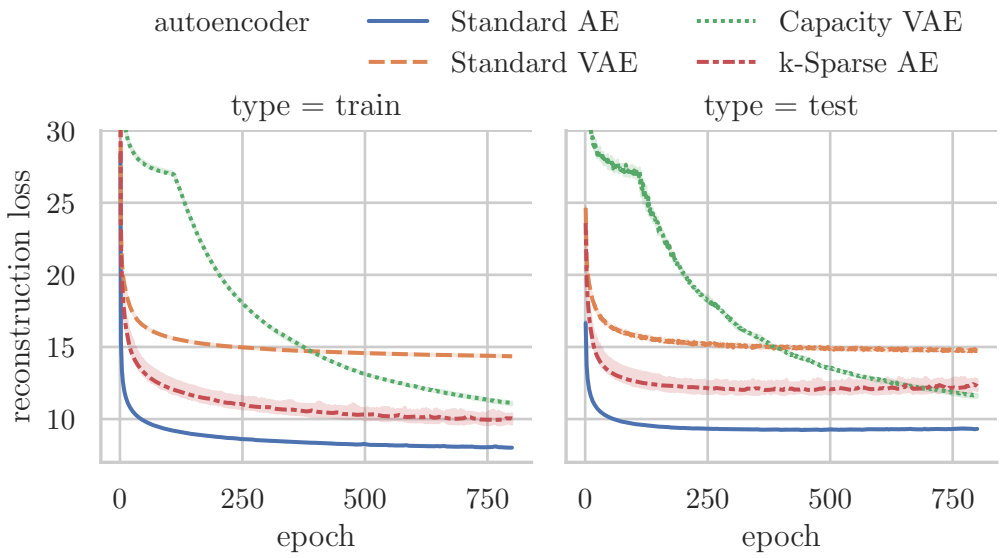

Figure 3.4: Plot of the reconstruction loss on the Fashion-MNIST dataset for the used AE types with fully-connected encoder and decoder.

AE implementation. Especially the high deviations between multiple runs of the k-sparse AE suggest an instability. The comparison between the standard VAE and the capacity-controlled VAE shows major differences. Until epoch 110 the capacity control is too restrictive and as a consequence, the reconstruction loss seems to plateau. After the capacity control value reached 110, the reconstruction loss improves again, even surpassing the standard VAE implementation. This suggests that the capacity control not only has a positive effect on the disentanglement but also on the reconstruction performance.

In addition to the AE type, the encoder-decoder combination is compared in Fig. 3.5 in terms of reconstruction loss. While the VAE with the *conv* encoder-decoder combination reaches the lowest test loss, it starts overfitting after epoch 400. This confirms the expectation that more parameters lead to overfitting. The Spatial Broadcast decoder is the worst performing encoder-decoder combination with the most deviation.

This section showed that all AE implementations work correctly. Their performance varies but most of them perform sufficiently. For the usage within the LEA, the k-sparse AE's high deviation is of concern. The *conv* encoder-decoder combination did show overfitting and for small input data reduces to a kind of fully-connected combination. The simple *fc* encoder-decoder combination is therefore mainly used for the further LEA analysis.

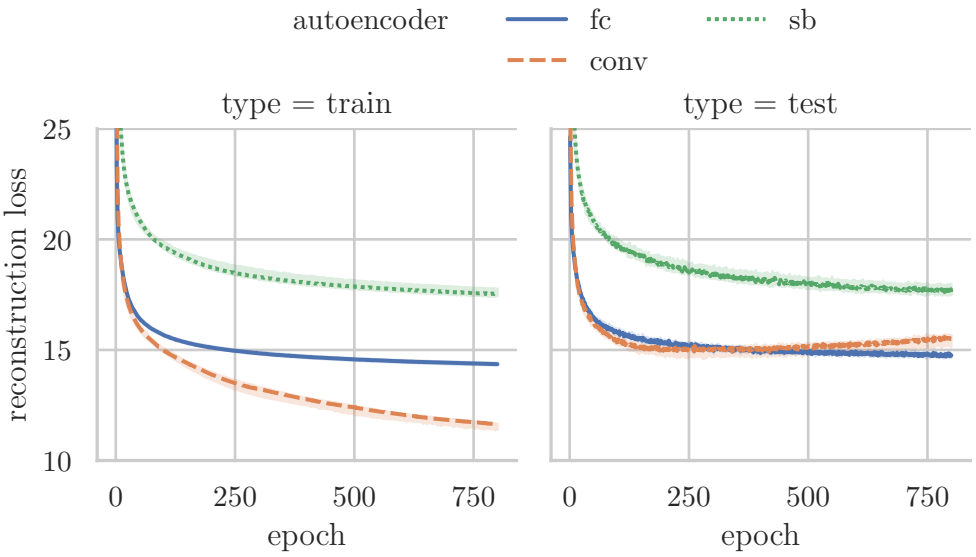

Figure 3.5: Plot of the reconstruction loss on the Fashion-MNIST dataset for the used encoder-decoder combinations with a standard VAE.

Chapter 4

Method

This chapter introduces the methods with which the quality of the experiments, the analysis, and the results presented in this work is ensured.

4.1 Results Reliability

This work uses various metrics to evaluate and validate all experiments and results. For all datasets used in this work, a split as defined in Section 4.8 into training and test is applied. All metrics are evaluated on the training and test data after every epoch. Metrics are recorded as an average of all batch results. For the training data, this is done during training and not as an additional step. The metric results for the training phase, therefore, incorporate results from the start of this phase. Based on this evaluation detail, the training results can look slightly worse than the testing results.

The majority of reported results in this work are based on 10 independent runs. Result plots in this work that display metrics show per epoch the median of all 10 metric results. Besides, the deviations of all runs are displayed as background hue. This gives a clear image of the median performance, the repeatability of these results, and the stability of the models. If not stated explicitly otherwise, this applies to all results shown in this work.

While images are taken from single runs, they are representative of results achieved by the used hyperparameters, respectively. If not specified otherwise, images are based on the test set. Most images presented in this work have been scaled up due to their low resolution.

4.2 Reconstruction Metrics

For an AE the reconstruction performance is most important. If the reconstruction process does not work, there is no guarantee that the encoding process works. Vice versa, if the reconstruction is possible, the code needs to contain the necessary information and hence the encoding process is successful. The same is true for the LEA and its window reconstructions. Since a LEA only forwards the code maps to

subsequent layers, it is of utmost importance that the code contains the necessary information. This work assumes this requirement to be fulfilled if the reconstruction is successful. Therefore it is important to measure and monitor the reconstruction performance with different metrics.

The reconstruction loss is monitored per LEA in a network. It is important to note that the reconstruction loss only refers to the squared distances between inputs and reconstructions. While the AE loss of different types is not comparable, the reconstruction loss is. Since the LEA can be used after any layer of the network, the input values depend on the previous layers. The input values are therefore unknown and can constantly shift. This behavior leads to a misleading reconstruction loss. The observed phenomenon is that a change in the input value range directly affects the reconstruction loss. For example, when the input values increase, the distance also increases. This is the case, even when the relative distance between both values stays the same. Therefore, the reconstruction loss can appear to be increasing while relative to the input value range the reconstruction loss is decreasing. To counter this misleading effect, a normalization of the reconstruction loss is introduced. The main idea of this normalization is to scale the reconstruction loss i.e. the difference between input and reconstruction by the maximum-to-minimum differences of input values. This reconstruction loss normalization is defined in Eq. (4.1) as $RLN(w, r)$.

$$RLN(w,r) = \frac{1}{l \cdot \#w} \sum_{i=1}^{l} \sum_{u=1}^{\#w} \left(\frac{|r_{i,u} - w_{i,u}|}{|min(w) - max(w)|} \right)^2 \quad \text{where } l = b \cdot W \tag{4.1}$$

The maximum-to-minimum differences in window values are defined as $|min(w) - max(w)|$. Compared to the reconstruction loss defined in Eq. (3.2), the mean of all values is calculated. The normalized reconstruction loss is only used for the validation and not directly for the optimization.

The effect outlined above can be seen in Fig. 4.1, where the raw reconstruction loss is plotted in blue and the normalization in orange. While the reconstruction loss seemingly increases, the normalization decreases and thereby shows a correct representation of the training progress. Tests performed during this work are therefore only compared in terms of their normalized reconstruction loss. The normalization also avoids reconstruction loss discrepancies based on the window size.

While the normalized reconstruction loss helps to visualize the training progress regarding reconstruction and to compare multiple runs with different setups, it does not directly give insights into the reconstruction quality. Therefore, the reconstructions are visually compared against the true input image. This comparison can be done on a per-window basis or for the whole input image.

For larger windows, the per-window comparison clearly shows exact differences in the reconstruction compared to the input image. Since for smaller images, the details are too fine and visually not easily validatable, the per window comparison is

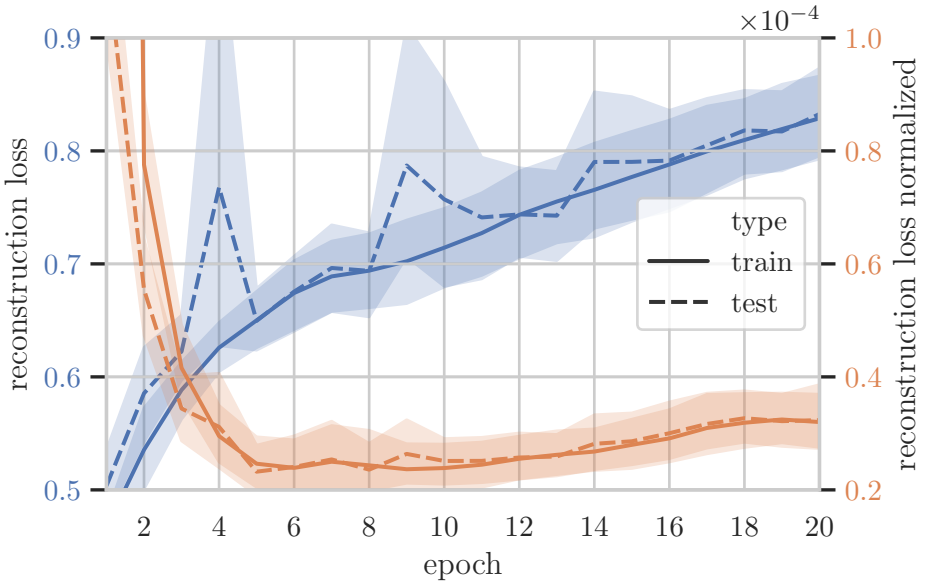

Figure 4.1: Comparison of raw and based on input value range normalized reconstruction loss on the dSprites dataset. A LEA (VAE - *fc*) following a convolutional layer.

not as useful. It is used as an additional help when inspecting unexpected behaviors.

More useful is the recombination of all reconstruction windows to a whole image. This allows a comparison between the whole original input and the whole reconstruction and is, therefore, easier to validate. The recombination of windows is done by performing an inverse of the windowing function.

The reconstruction value distribution usually differs from the input value distribution. To link reconstructions to their true values and to track them over the training course, they are combined to one image. For the combination both images have to be in the same value range or else the color space of the images shifts. Hence, outliers in the reconstruction lead to grayish comparison images which are again not easily validatable. This is solved by simply clipping the reconstruction value range to the minimum and maximum of the input values. Thereby outliers are ignored and the comparison images are visualized in a proper color range, i.e. black to white for every channel. These clipped reconstructions are also only used for the validation and not for the optimization. Since the clipping ignores the value distribution, it is helpful to also compare the value distributions of the reconstruction and the input. For this reason, the histograms of both, the reconstruction values and the input values, are visually compared over the course of training.

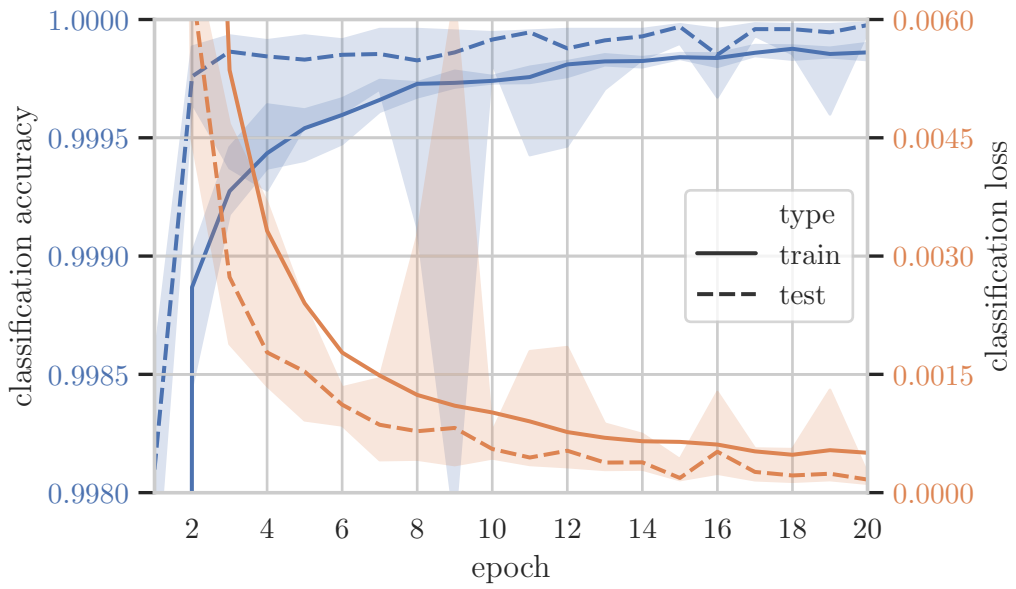

Figure 4.2: Median of classification accuracy (blue) and loss (orange) of the shape classification on the dSprites dataset

4.3 Classification Metrics

The network incorporating LEAs has to optimize two tasks — the encoding-decoding task and the main task. The main task only and directly makes use of the code. The classification metrics, therefore, verify that the code contains the necessary information for the main task. These metrics are also used to compare different LEA configurations with each other or against a LEA free baseline. This comparison focuses on the generalization performance.

The generalization performance of the classification is measured by the accuracy and the log loss. As shown in Fig. 4.2, an increasing and generally higher accuracy as well as a decreasing and generally lower loss indicates a better performance. The log loss was introduced and defined in Eq. (2.21). The accuracy metric measures the fraction of correct classifications and in this work is defined as in Eq. (4.2).

$$ACC(x,y) = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1 & \text{if } arg\,max_u\,(t_{i,u}) = arg\,max_u\,(x_{y,u}) \\ 0 & \text{else}\end{cases} \tag{4.2}$$

This formulation assumes a one-hot encoding of the target samples t, i.e. every class is a separate value with values of $\{0,1\}$. Correct classification is given if the highest value $y_{i,u}$ in the output y_i marks the correct class, that is, the highest value of y and t are at the same index u.

Both the loss and the accuracy, are monitored in combination to detect overfit-

ting issues. Overfitting can be caused by the model complexity, a class-imbalance, overconfidence, and false correlations in the training data (Poole and Mackworth, 2017). This work follows the standard procedure to detect overfitting by validating the performance on a separate test set of unseen data. One type of overfitting, overconfidence, is an issue that is present with models tested as part of this work. On the one hand, a stable accuracy with an improving loss is a sign for a positive increase in confidence. On the other hand, a stable accuracy with an increasing loss is a sign for a negative confidence effect, namely overconfidence. This can be due to a view wrong predictions that have little effect on the accuracy but an unproportionate huge effect on the loss. In the case that for results of this work only the accuracy is shown, the loss and accuracy show the same behavior and do not show any other effects. The accuracy is more useful to judge the general performance since it is always between $[0, 1]$.

Using the classification metrics for the hyperparameter selection leads to a pitfall. Selecting configurations that perform well for the classification objective seems to introduce a bias of selecting models with a worse reconstruction performance. This is not generally true, therefore it is important to always validate the classification and reconstruction metrics.

4.4 Reconstruction Confidence Map

The LEA performs a per-window encoding and reconstruction. The reconstruction loss can be defined per pixel, per window, or per batch. Compared to the per batch reconstruction loss as shown in Eq. (3.2), the only difference to the per window reconstruction loss as shown in Eq. (4.3) is that the reconstruction loss is only calculated for every window, indexed by the parameter i.

$$RL\left(w, r, i\right) = \sum_{u=1}^{\#w} (r_{i,u} - w_{i,u})^2 \tag{4.3}$$

With respect to all reconstruction losses of all batches, the performance of a window reconstruction can be formulated as confidence in the reconstruction of this window. The confidence is defined in Eq. (4.4) as the inverse of the min-max normalized reconstruction loss of the corresponding window.

$$CONFI(w, r, i) = 1 - \frac{RL\left(w, r, i\right)}{max_u\left(RL\left(w, r, u\right)\right)} \tag{4.4}$$

The minimum for the min-max normalization is set to zero.

Similar to the recombination of per-window codes to code maps, it is also possible to recombine the per-window reconstruction confidences to a confidence map. This

visualization highlights the reconstruction performance of the underlying AE on parts of the input image. This is a good indicator of potential encoding-decoding issues as well as for easy and challenging encoding-decoding regions in the input data. Monitoring this visualization over the training course gives insights into the learning progress and evolution of the AE. The evolution of the reconstruction confidence map over multiple epochs is shown in Fig. 4.3.

4.5 Code Maps

As laid out in Chapter 3, code maps are the output of a LEA. More specifically, code maps are the recombination of codes that resulted from a windowing and per-window encoding process. The code maps can be viewed as a new output image, where the channel dimension references the code maps. A channel, therefore, refers to a code of the encoder or rather a grayscale image visualizing this single code for all windows per input image. An example of the visualization of these code maps is shown in Fig. 4.4.

Visualizing these code maps gives insights into the encoding process or more specifically what single code variables encode. To be precise, these code maps give insights into the state of entanglement and disentanglement of the code, respectively. Highly entangled codes will result in noisy code map images without any obvious human interpretable correlations. Thus the code maps are only intended for the observation of disentanglement.

Important to note is that using the code maps for the hyperparameter selection leads to a bias towards visually pleasing code maps. This might not only result in worse classification and reconstruction performance but might also hinder a better encoding or even better disentanglement. These code maps should therefore only be used to gain insights and not as a direct performance metric.

4.6 Robustness Metrics

The robustness of the models is measured with and against adversarial attacks quantified by perturbation distance and the success rate. To compare the performance for a diverse set of adversarial attacks the Python package Foolbox[1] is used. Foolbox (Rauber *et al.*, 2017) is a toolkit that provides implementations of gradient-based, score-based, and decision-based adversarial attacks. A subset of these attacks is used to evaluate the robustness of the models. This subset is selected to test a diverse set of attacks with low runtime.

From the gradient-based attacks the Gradient Sign (Goodfellow *et al.*, 2014), the Projected Gradient Descent (Madry *et al.*, 2017), the Deep Fool (Moosavi-Dezfooli

[1]Bethge Lab, Foolbox Python package: github.com/bethgelab/foolbox (used version: 1.8.0)

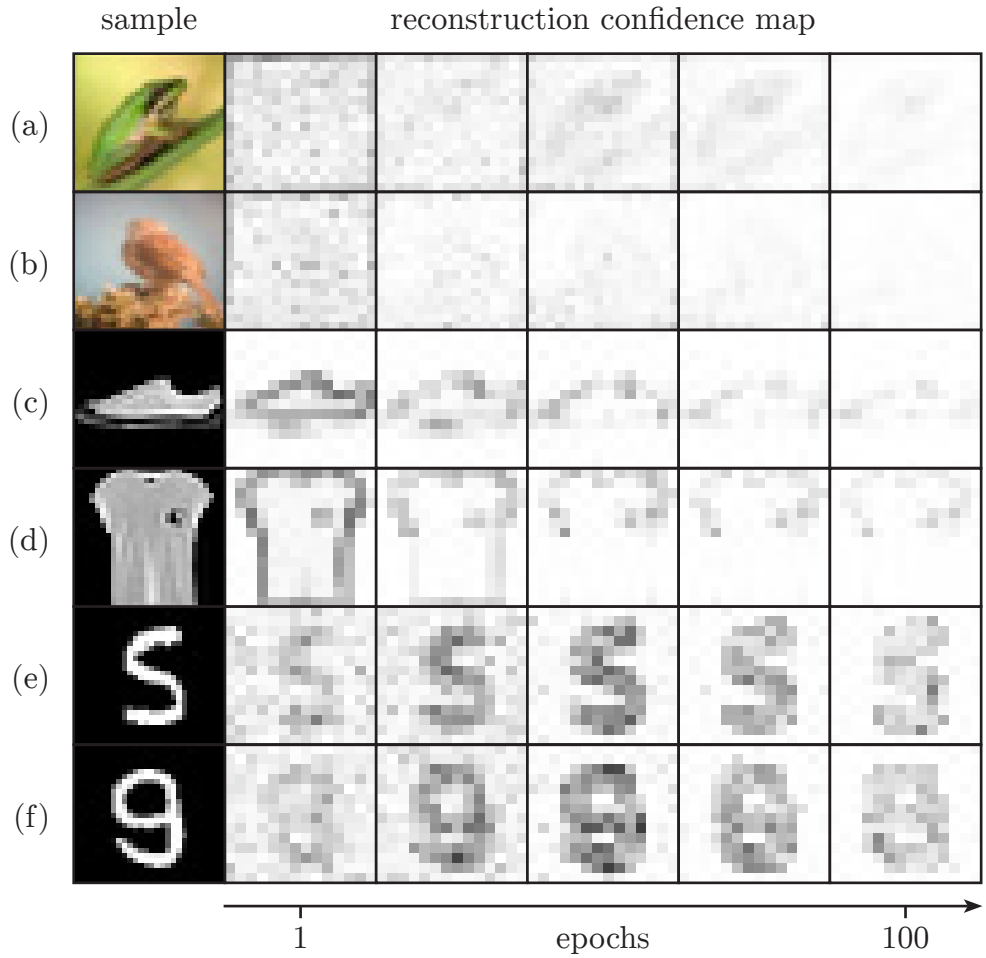

Figure 4.3: Reconstruction confidence map over the course of 100 epochs. White shows areas where the autoencoder is confident and black shows the unconfident areas. Samples are from (a, b) CIFAR-10 trained with a capacity VAE - *fc*, (c, d) Fashion-MNIST trained with a standard AE - *fc*, and (e, f) MNIST trained with a capacity VAE - *fc*

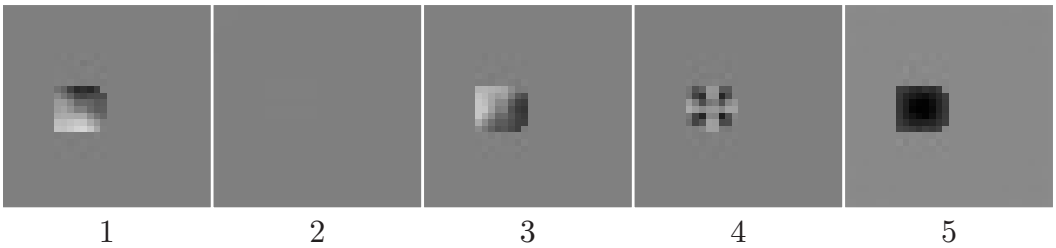

Figure 4.4: Code maps of the first LEA in a CNN trained on the dSprites dataset with a window of 4×4 px and 5 codes. Since every code shows a different and meaningful pattern, a high degree of disentanglement can be assumed.

et al., 2015), the ADef (Alaifari *et al.*, 2018), the SLSQP, and the Saliency Map (Papernot *et al.*, 2016) attacks are selected. For example, the Projected Gradient Descent attack searches for a minimal perturbation by computing the gradient for the model making a misclassification while restricting the maximum perturbation distance.

From the score-based attacks, the Single Pixel (Narodytska and Kasiviswanathan, 2016) and the Local Search (Narodytska and Kasiviswanathan, 2016) attacks are selected. The Single Pixel attack switches single pixels to either black or white until an adversarial example is found. The Local Search attack first searches for the pixel with the highest influence on the classification, then modifies this pixel and repeats this process until an adversarial example is found.

From the decision-based attack the Spatial (Engstrom *et al.*, 2017), the Gaussian Blur, the Contrast Reduction, the Additive Gaussian Noise, and the Salt and Pepper Noise attacks are selected. The Spatial attack performs translations and rotations until an adversarial example is found. The Gaussian Blur attack increases the blur until an adversarial example is found. The Contrast Reduction attack reduces the contrast until an adversarial is found. The other attacks work accordingly.

For all attacks, the default implementation parameters are used. The attacks are untargeted with the goal of misclassification, i.e. all attacks try to find an adversarial example that leads to a classification that does not match the label. Correct classification is defined by the accuracy metric as shown in Eq. (4.2).

The perturbation distance is measured in terms of the MSE, between input and the successful adversarial example. To ensure independents to the input value scale, both inputs are min-max normalized to $[0, 1]$ (Rauber *et al.*, 2017). Only considering successful perturbations for the mean distance, makes it independent to the success rate.

The success rate measures the fraction of successful perturbations. A perturbation for a sample is successful if an adversarial attack could find an adversarial example that led to a misclassification of this sample.

The robustness of multiple models against multiple adversarial attacks is visualized with a heatmap. The columns of this heatmap contain the attacks and the rows the models. This visualization compares baselines against models with LEAs as well as different hyperparameter configurations with respect to their robustness per attack. Both, the perturbation distance and the success rate are shown in the same heatmap. For example, such a heatmap is shown in Fig. 4.5. Darker colors visualize a higher perturbation distance, meaning that stronger modifications were necessary to find an adversarial example. Smaller circles visualize a lower success rate, meaning that the attack did not find an adversarial example more frequently. Therefore, smaller circles and darker colors show higher robustness against the respective attack. Different attack results for one model are not comparable with respect to the model. A comparison between these results only informs about attack

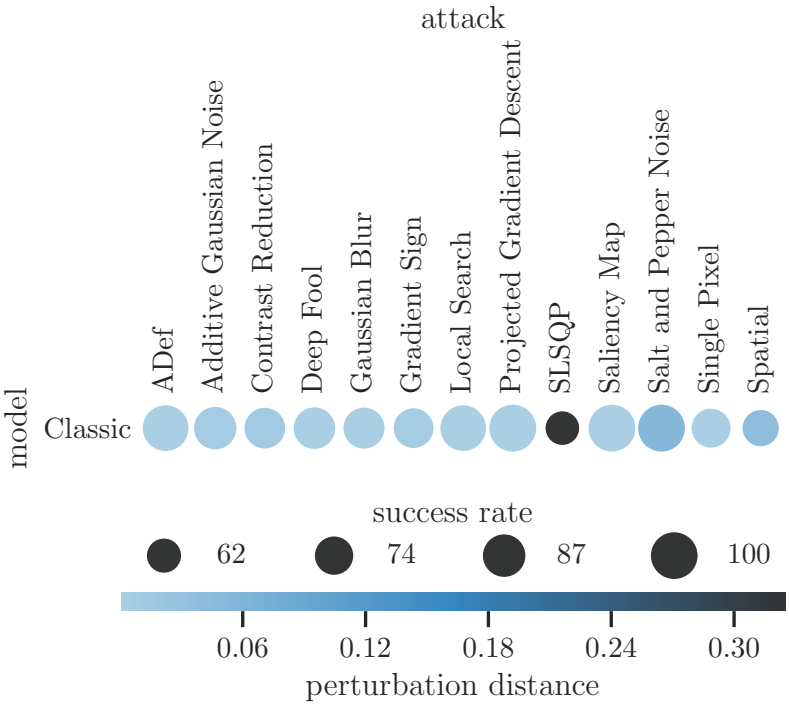

Figure 4.5: The heatmap of the perturbation distance and success rate comparison for all models and all attacks is similar to this example that only contains the Classic baseline evaluated on the dSprites dataset. Smaller and darker circles are better.

differences but not how the model performs in comparison to different attacks. This information comes only from the comparison between multiple evaluated models and especially the baseline.

When colorizing based on the perturbation distance, the difference between rows is not visible since single attacks that inherently result in a higher perturbation distance overshadow the other attacks. This effect is noticeable in Fig. 4.5 for the SLSQP attack. As a consequence, heatmaps shown in this work are standardized using the respective baseline. Meaning, per attack the perturbation distance and success rate is normalized using the standardization as defined in Eq. (2.14) with the respective baseline value as mean μ. The standard deviation σ in the standardization also uses the baseline value as mean. As a result, the baseline marks the center with zero. For the success rate comparison, negative values (i.e. smaller circles) indicate better robustness. For the perturbation distance comparison, a color space from red to gray and from gray to blue is used. Better robustness in terms of perturbation distance is marked by positive values (i.e. blue colors) and vice versa.

Running these attacks can take a considerable amount of time. For the selected attacks this varies on average from 86 ms for the ADef attack to 7 s for the SLSQP attack. The robustness on all attacks is therefore only evaluated on 400 samples from the test set. More precisely, the first 20 samples of the 20 first batches are used for this evaluation. The samples and batches are always in the same order for the evaluation runs of every model.

4.7 Baselines

Two types of performance comparisons are performed in this work. The first being comparisons between hyperparameter configurations of models using LEAs. The second being comparisons between models with LEAs and models without LEAs. Models which do not contain a LEA are referred to as baselines.

For separate analyses, three different baselines are used. In general, these baselines are used to analyze the effect and potential advantages of using a LEA. One of those three baselines is called Classic, one is called CNN-POOL and one is called Decoder-free.

The Classic baseline represents a simple classical CNN architecture with convolutional (*conv*), pooling (*pool*), and fully-connected (*fc*) layers. Specifically, this model consists of five layers with the sequence *conv-pool-conv-pool-fc-fc*. The convolutional layers perform a stride of one and the pooling layers a stride of two. The pooling layers have a window size of two, equal to their stride. The first convolutional layer has a window size of five and the second convolutional layer a window size of three. This model works well on commonly used image datasets but does not represent state-of-the-art. For a comparison, it is possible to replace pooling

layers by LEAs. The Classic baseline with LEAs replacing pooling operations is called CNN-LEA. In the CNN-LEA model, alternate training is performed. This is why the convolutional layers have a stride of one and the pooling or LEA performs the actual striding. This comparison should show the effect of applying the LEA in a classical CNN architecture on the classification performance.

The POOL-FC baseline uses only one pooling layer and fully-connected layers, i.e. the layer sequence *pool-fc-fc*. The main focus lays on the single pooling layer. Only one pooling layer is used to allow testing with larger strides. The effect of specifically replacing pooling with LEAs on the classification performance while using the same hyperparameters is analyzed using this baseline. The POOL-FC baseline with LEAs as pooling replacements is called LEA-FC. In the LEA-FC model also an alternate training is performed.

The LEA-only model is mainly a LEA only architecture, with a layer of LEAs and therefore a *lea-fc-fc* layer sequence. Training in this setup is generally performed as a joint training of the main objective and the reconstruction objective instead of an alternate training. The Decoder-free baseline uses the LEA-only architecture but only with respect to the main objective. The reconstruction objective is ignored and the decoder path of the architecture is never used. Comparing the Decoder-free baseline against the LEA-only model should reveal the effects of the reconstruction objective, i.e. the regularization of the LEA output.

The final fully-connected layers in all of these models are the same. The first fully-connected layer uses 1024 neurons and the second fully-connected layer maps these onto the output target. It is important to note that none of the above models compares against state-of-the-art results. They are meant to analyze effects and not to show new performance achievements.

4.8 Datasets and Data Preprocessing

Experiments and analysis were performed using four image datasets — MNIST, Fashion-MNIST, dSprites, and CIFAR-10. The MNIST dataset (Lecun *et al.*, 1998) contains grayscale images of ten handwritten digits. These are single digits in the range of $[0, 9]$. The Fashion-MNIST dataset (Xiao *et al.*, 2017) contains grayscale images of ten fashion articles. These fashion articles are T-shirts / tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. The dSprites dataset (Matthey *et al.*, 2017) contains black and white images of three shapes with four known spatial transformations. The shape is either a square, an ellipse, or a heart and images vary in the scale, orientation, and position of these shapes. The CIFAR-10 dataset (Krizhevsky, 2009) contains colored images of four vehicles types and six animals. The four vehicles are airplanes, automobiles, ships, and trucks and the six animals are birds, cats, deers, dogs, frogs, and horses.

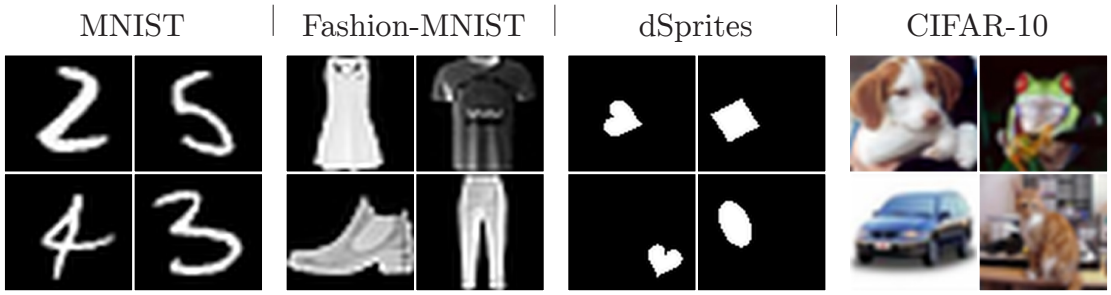

Figure 4.6: Example images of the used datasets — MNIST, Fashion-MNIST, dSprites, and CIFAR-10

The datasets, MNIST[2], Fashion-MNIST[3], dSprites[4], and CIFAR-10[5], are available online. A set of four example images per dataset is shown in Fig. 4.6.

For the data pipeline implemented in this work, all dataset features and labels need to have the same format. Features, i.e. the input images, need to have a value range of $[0, 1]$ and a channel last data format. Labels, i.e. the target output classes, need to be one-hot encoded, where every class is a separate value with values of $\{0, 1\}$. Since this is not necessarily the case, the data pipeline incorporates some preprocessing steps. The image might come in an 8-bit integer format with values in the range of $[0, 255]$. In this case, the images are scaled to the desired values range with a min-max normalization, i.e. a division by 255. For images with a channel first format, the channel dimension is swapped to achieve a channel last format. On all dataset labels, a one-hot encoding is performed. Before training, all training samples are shuffled to ensure a stochastic training process. This shuffling and all other random processes used during the data preprocessing are initialized with a fixed seed to ensure comparability by repeatability.

While all other datasets come with a predefined test set, the dSprites dataset only comes with 737 280 labeled images. These images come from all permutations of three shapes, six scales, 40 orientations, 32 x-positions, and 32 y-positions. To have similar training and testing set sizes as given by MNIST and Fashion-MNIST as shown in Table 4.1, about 70 000 samples are selected randomly from the dSprites dataset and randomly split into a training set with a fraction of about 85.8 %. As shown in Table 4.1, this random process with fixed seeds results in a subset of 69 978 samples, which is about 9.49 % of the whole dataset, with 59 972 training and 10 006 testing samples. The dataset size approximately reduces from 18.87 MB to around 1.79 MB.

The main advantage of the MNIST, Fashion-MNIST, and CIFAR-10 is the fast

[2]MNIST dataset obtained from: yann.lecun.com/exdb/mnist/
[3]Fashion-MNIST dataset obtained from: github.com/zalandoresearch/fashion-mnist
[4]dSprites dataset obtained from: github.com/deepmind/dsprites-dataset
[5]CIFAR-10 dataset obtained from: cs.toronto.edu/~kriz/cifar.html

Table 4.1: Comparisons in terms of image size, sample count and file size of the used datasets — MNIST, Fashion-MNIST, dSprites, and CIFAR-10

Dataset	Image size	Training samples	Testing samples	Total samples	File size
MNIST	28×28×1 px	60 000	10 000	70 000	11.29 MB
Fashion-MNIST	28×28×1 px	60 000	10 000	70 000	30.12 MB
dSprites	64×64×1 px	59 972	10 006	69 978	∼1.79 MB
CIFAR-10	32×32×3 px	50 000	10 000	60 000	181.85 MB

runtime for experiments. MNIST and Fashion-MNIST are equal in terms of their dataset structure and sample format. Fashion-MNIST has a higher classification complexity and is, therefore, more meaningful for experiments (Xiao *et al.*, 2017). The images of the MNIST, Fashion-MNIST, and dSprites datasets show simple structures. As a consequence, models trained on these datasets are more interpretable. Since the generative factors of the dSprites dataset are known, it is used to analyze disentanglement effects.

Based on the fast runtime, the easy interpretability, and the existing task complexity, Fashion-MNIST is used for most experiments during this work. For all experiments related to the disentanglement property, the dSprites dataset was used. MNIST and CIFAR-10 were used for additional validations. Results shown in this work, are from any of these four datasets. For every graphic presented in this work the used dataset is specified in the respective caption.

4.9 Environment

The implementation accompanying this work was implemented with the Python[6] programming language and the ML framework TensorFlow[7]. The models were trained on the computer cluster of the Training Center for Machine Learning project of the University of Tübingen funded by the Federal Ministry of Education and Research. The cluster provides 40 compute nodes where each is equipped with a 2 TB solid-state drive, 256 GB random-access memory (RAM), an Intel XEON E5-2650 v4 central processing unit (CPU), and four Nvidia GeForce GTX 1080 Ti graphics processing units (GPUs). Each training run was configured to use one node with four CPU threads, 6 GB RAM per thread and one of the four GPUs. During training these resources were reserved, therefore there should be no interference from other tasks on the same node.

[6]Python Software Foundation, Python programming language: python.org (used version: 3.6.5)
[7]Google LLC, TensorFlow framework: tensorflow.org (used version: 1.3.1)

Chapter 5

Analysis

This chapter focuses on the analysis of different aspects, experiments, and theories regarding the LEA.

5.1 Unsupervised Learning of Local Features

Classical ANN approaches to object recognition are often most concerned with shape and texture of objects to achieve classification. The human visual processing can be described in stage, where the earliest stages detect basic features like edges, contrasts, and orientations and later stages group these features into higher-order objects with depth information and separations between object and background (Ward, 2015). Since we can view our world in a hierarchical structure of always finer subobjects and a whole object only makes sense if the required subobjects are present, it makes sense to discriminate subobjects and to discriminate higher-level objects based on these. While current ANNs probably implicitly do something similar, they do not explicitly combine subobjects over multiple stages. From the perspective of image data this notion of subobjects clearly is defined by the window size. The notion of grouping can be achieved by subsequent ANN layers. This forms one of the core motivations behind the LEA.

Training ANNs via supervision, i.e. by providing class labels for the object recognition task, is most common. This reveals the main issue for implementing an approach with explicit detection and combination of subobjects — the subobjects are unknown and therefore supervised training is impossible. With deep bag-of-features models (BagNets) Brendel and Bethge (2019) presented an approach with a similar kind of supervised training. The BagNets are trained to classify the target labels on smaller windows, thereby learning class-specific features. These local classifications are then recombined to a heatmap per class. This recombination process is similar to the recombination of codes in the LEA. As a final step, the sum over every heatmap yields a classification score for every class. In (Brendel and Bethge, 2019) promising results for the application of this approach on the ImageNet[1] dataset (Russakovsky *et al.*, 2015) are shown.

[1]ImageNet dataset can be obtained from: image-net.org

In the BagNets approach, the explanatory factors are still unknown. An imaginary dataset could contain class labels for every subobject of any degree of detail. It is unfeasible to create such a dataset by manually labeling windows of varying size. There might even be explanatory factors that even humans do not explicitly recognize. Therefore a learning process that automatically detects these explanatory factors without supervision must be used — unsupervised learning.

In the LEA the process of encoding windows is supposed to represent the unsupervised classification for explanatory factors. Important to note here is that a standard AE does not explicitly learn any classification like code. To reiterate, an AE needs to learn the features that most define the input data to fulfilling the reconstruction objective. While it is reasonable to expect these features to correlate with the explanatory factors a human would recognize, these might be present in an intangible way. In other words, for classification, an indicator per class is expected, but all code variables might not have any kind of separation. Therefore it is intangible for humans to understand and judge the classification capabilities of a fully trained standard AE. The missing separation in the code is referred to as the code being entangled.

In contrast, if the learned classes are separate, then the unsupervised learning is successful if for humans it is easy to identify the distinct classes from the model output, i.e. the code. In the LEA this means that the code maps show interpretable and meaningful images. To achieve this, this work uses disentanglement techniques. Even if this process works, it is not clear if ANNs benefit from these human interpretable representations.

While the standard and k-sparse AE produce highly entangled code representations, VAE-based models show some form of disentanglement. Especially the KL divergence regularization is important to obtain a disentangled code (Higgins *et al.*, 2017). The capacity-controlled VAE (Burgess *et al.*, 2018) showed the most promising results and is, therefore, the preferred VAE method. Another recent technique that showed promising results is the Spatial Broadcast decoder (Watters *et al.*, 2019). This decoder can be used in combination with any AE type. To verify the disentanglement properties of these two techniques, they are directly tested on the dSprites dataset. Important to note here that for this test only the AE implementations are used without the LEA specifics. The dSprites dataset is a generated dataset that has an important property for the test of disentanglement — all generative factors are known. Namely, the generative factors are the shape, the scale, the orientation, the x-position, and the y-position. It makes sense that good codes for these images have to consist of these factors. Consequently, it can be assumed that a perfectly disentangled code would map one factor onto one code variable. Therefore, when using this AE for image generation, changing a code variable should only change the respective factor in the generated image. For this test, the capacity-controlled VAE was combined with the Spatial Broadcast decoder and fully trained on the whole dSprites dataset. To independently modify

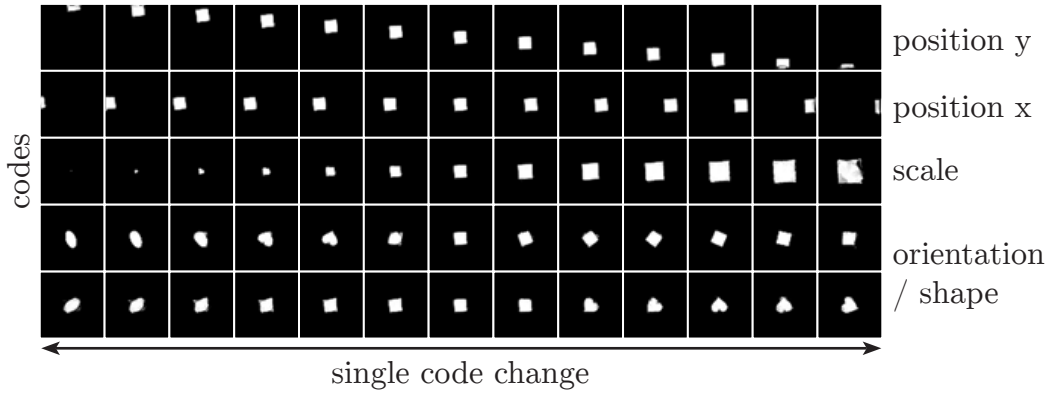

Figure 5.1: Generated images resulting from single code changes while all other code variables remain at zero. Only five of 10 code variables encode information and are shown here. The shown images are the result of changing the respective codes in the ranges $[-3, +3]$ with 0.5 steps for the x and y position, $[-2.1, +2.1]$ with 0.35 steps for the scale, $[-1.2, +1.2]$ with 0.2 steps for the first orientation / shape code, and $[-1.1, +0.7]$ with 0.15 steps for the second. The autoencoder was trained on the dSprites dataset.

code variables a small user interface (UI) that regenerates images on code change was implemented. This UI was used to modify codes for the fully trained AE of this test. The resulting images of changing these codes in limited ranges around zero are shown in Fig. 5.1. The training was performed with 10 codes but only five learned to encode something. Changing the five other codes does not result in any notable change in the generated image. Therefore only the result images for the modification of the five codes that learned to encode something are shown. As can be seen, this unsupervised training resulted in a well-disentangled code. Especially the positions and scale factors are perfectly separated. The orientation and shape are entangled over two codes.

These results prove that it is possible to discriminate explanatory factors by learning disentangled and meaningful codes without supervision. Instead of doing this on whole input images, the goal of this work is to learn disentangled codes of windows. For this purpose, a test setup is prepared to investigate the disentanglement of the LEA codes. A LEA is used directly on the raw dataset images with only the local reconstruction objectives. This eliminates side effects of the convolutional layers, the pooling layers or the classification objective. The windowing is done for 4×4 px and as AE the capacity-controlled VAE with the fully-connected encoder-decoder variant is used. The target code consists of 10 values. To get a higher resolution, a stride of one was used. This setup is run on all four datasets, i.e. CIFAR-10, dSprites, Fashion-MNIST, and MNIST. The resulting code maps with the respective input image can be seen in Fig. 5.2. Note that for all four datasets the reconstruction performance is good.

These code maps do not seem to show any resemblance to any obvious subclasses.

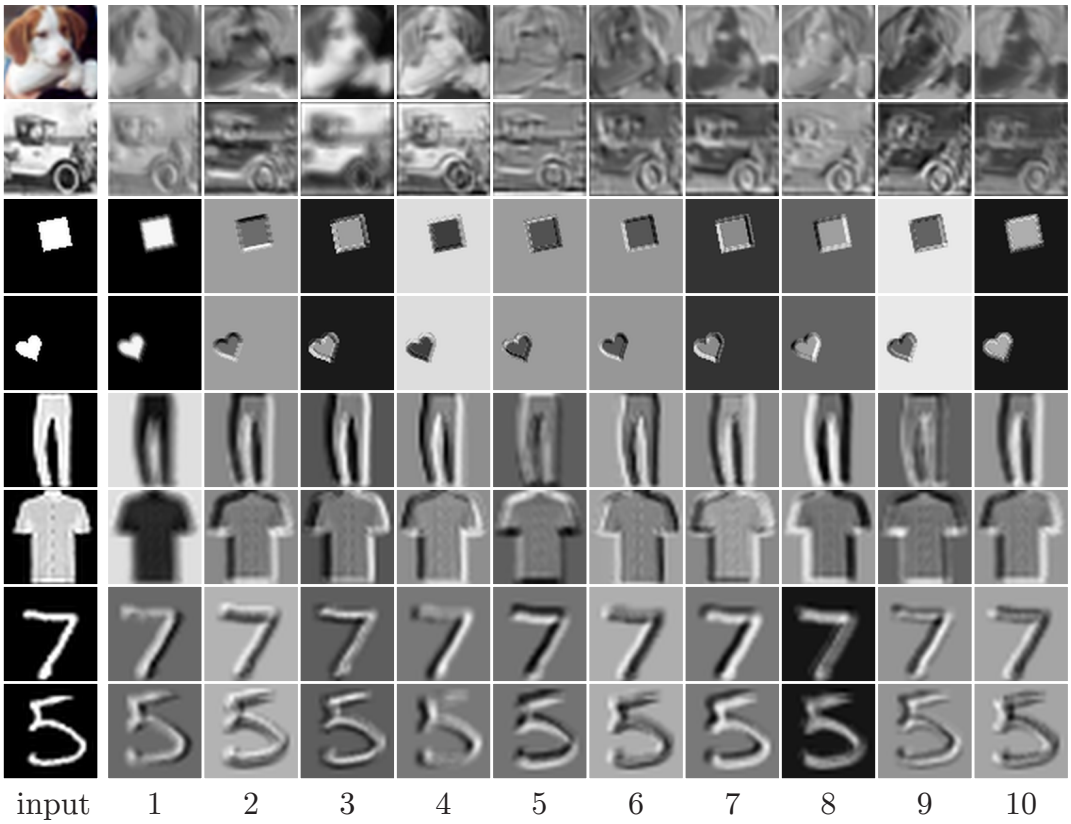

Figure 5.2: On the right 10 code maps are shown, which are the result of a LEA directly trained on the input images displayed in the first column. All four datasets were used and per dataset two results are shown. From the top to the bottom these are CIFAR-10, dSprites, Fashion-MNIST, and MNIST. Since the LEA window size was 4×4 px, mostly edge related features are learned.

The used datasets are of relatively low resolution and especially the dSprites, Fashion-MNIST, and MNIST datasets do not particularly contain subclasses. The chosen window size of 4×4 px only allows the AE to learn edge related codes. That every code encodes properties for edges is especially visible in the Fashion-MNIST and MNIST results. Only the CIFAR-10 results show more diverse code patterns. While these results do not specifically show the unsupervised classification of subclasses, it is important to note that every code map shows a coherent pattern. Code maps, therefore, seem to encode specific features or situations. Further testing on these capabilities needs to be performed.

5.2 Equivariance to Spatial Transformations

Convolutional neural networks (CNNs) are known for their favorable properties in regards to invariance, especially their invariance to translations. If a CNN or more generally an ANN is trained on an objective that does not depend on spatial information, the assumption can be made that this unnecessary information is lost over the course of multiple layers. This loss in spatial information over multiple layers in such a network is what this work refers to as invariance to spatial transformations. In contrast, if an ANN would be specifically trained to preserve spatial information, the assumption can be made that this information is still present in subsequent layers. The further existence of this spatial information is what this work refers to as equivariance to spatial transformations. The reconstruction objective of an AE can be viewed as an objective that specifically preserves spatial information. A reconstruction is deemed successful if it is close to the original image and this is only the case if all spatial properties of the reconstruction are close to the original image as well. In this case, the code has to contain the spatial information and the AE is equivariant to spatial transformations. Since this holds for an AE and all codes are recombined without any value change, this also holds for a LEA.

This work proposes a method to test the preservation of spatial information, i.e. the equivariance to spatial transformations. This method tests the degree of equivariance in terms of the loss in spatial information when performing the operation under test. The loss in spatial information is called the spatial information loss. A trained model is tested by measuring the spatial information loss on every network layer. The more this loss decreases from layer to layer, the less this model or specific layers are equivariant to spatial transformations. The test result can be directly used to compare different models. To measure the spatial information loss, the ground truth spatial transformations need to be known. Since the dSprites dataset is mainly generated from spatial transformations, namely translation, rotation, and scale, it is a good choice for this test.

The test setup as described above and as used in this work is shown in Fig. 5.3. Images of the dSprites dataset are used as input x and the labels are used as

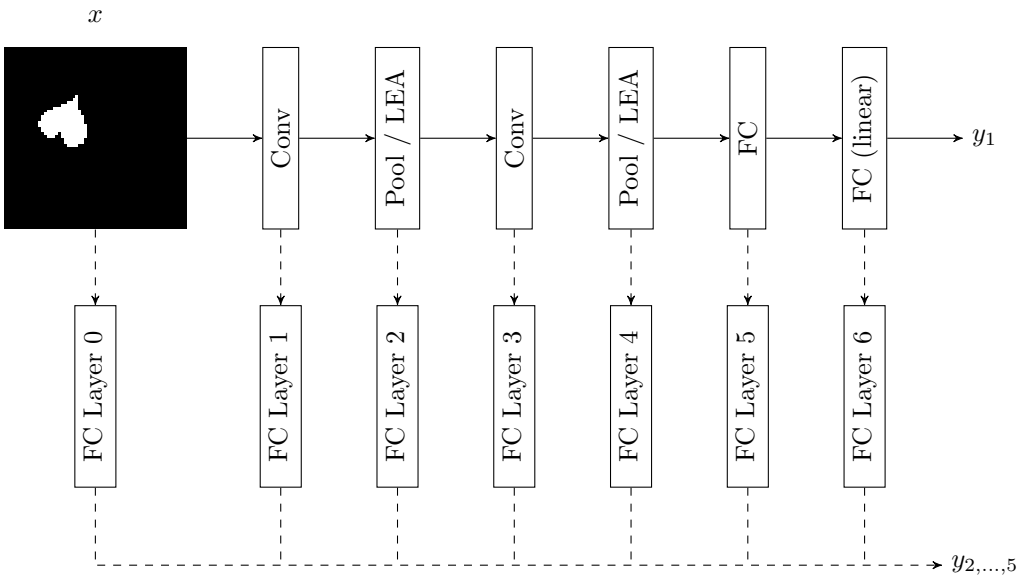

Figure 5.3: Model to evaluate the preservation of spatial information per layer. Solid lines show the pre-trained model (Classic / CNN-LEA). Dashed lines show the evaluation of every layer. x is an input image from the dSprites dataset. $y_{1,\dots,5}$ are the five classification targets: shape, scale, orientation, x-position, and y-position

target output y, where y_1 refers to the shape and $y_{2,\dots,5}$ to the scale, orientation, x-position, and y-position, respectively. To simulate an objective that does not have evident dependents on the spatial transformations, the network under test is fully trained with the objective to classify the shape of the sprites, i.e. if it is an ellipse, a heart, or a square. The network under test has no information about the spatial transformation labels. Every layer of the network under test, including the input image, is connected to a fully-connected layer with 1024 neurons. Each of these independent fully-connected layers uses a fully-connected layer with linear activation function per spatial transformation label and a log loss between the softmax-normalized output and label. In other words, every layer of the network under test is connected to a respective spatial information classifier. Every spatial classifier is trained until convergence while the network under test does not change due to fixed weights. For the network under test, Fig. 5.3 shows the Classic baseline compared to the CNN-LEA model.

The results of this test are shown in Fig. 5.4. These plots show the accuracy of classifying the correct spatial transformation for all four transformations.

The first insight comes from the difference between the different transformation types. While both translation transformations (i.e. change in x and y position) show the same behavior, the other two transformations, scale and rotation, show different behaviors. This indicates that the model handles different transformation types, translation, scale, and rotation, differently while not differentiating between

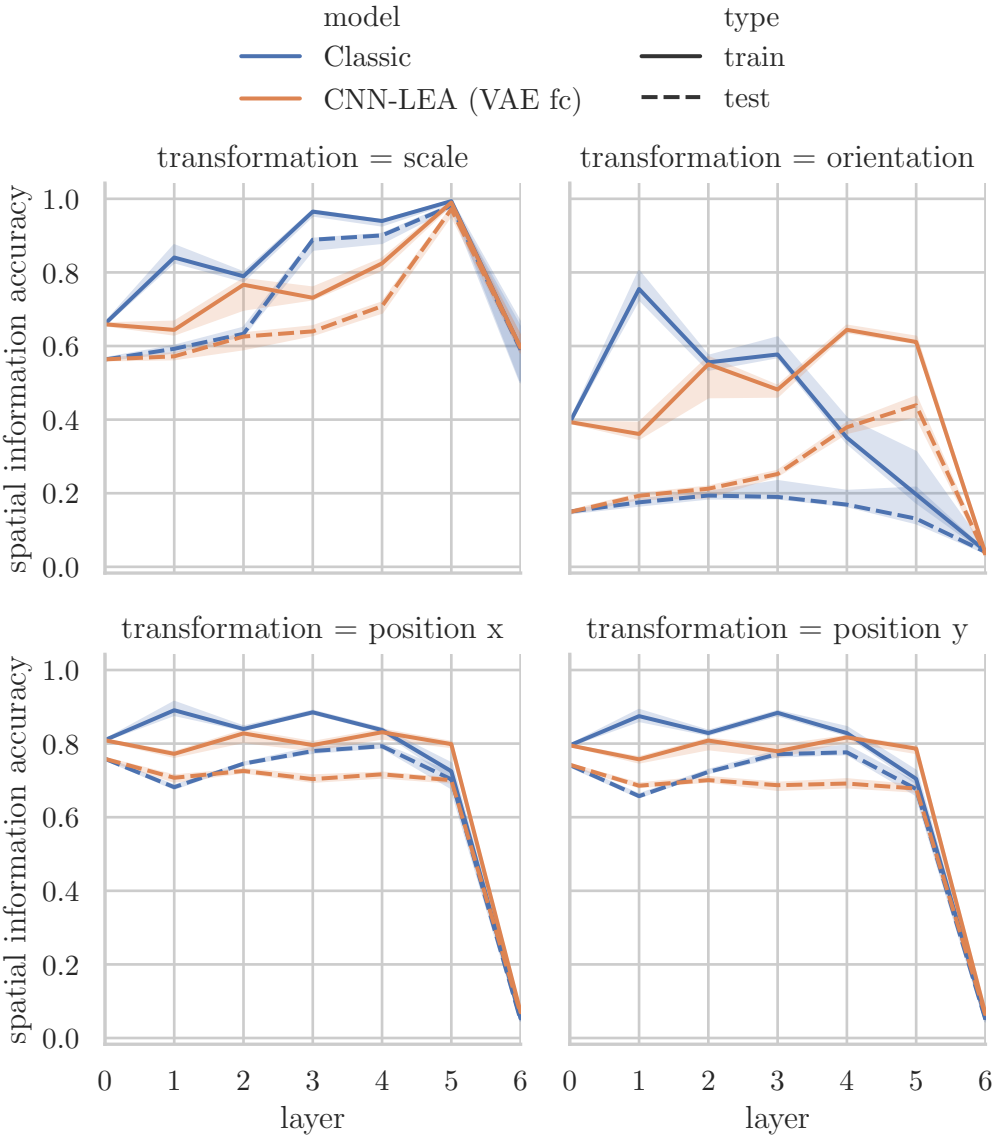

Figure 5.4: Accuracy of classifying the spatial transformations of the dSprites dataset, namely scale, orientation, x-position, and y-position, for every layer of the model under test.

transformations of the same type.

In general, an interesting insight from this evaluation is in regards to the last fully-connected layer with linear activation function, i.e. the output layer. For most transformations, layers, and models the spatial information is preserved by some extend. Except for the output layer, for all transformations and models, the output layer removes the spatial information. The expected output is the shape classification. This final output seems to only contain information about the shape and close to no information about transformations. This result contradicts the original hypothesis for CNNs being invariant to spatial transformations, in that the tested CNN did show equivariance to spatial transformations. Contrary to the original believe that spatial information is lost with every layer, the spatial information in most cases is preserved and potentially used throughout the network until it is dropped in the final layer.

Only analyzing the other layers except the output layer gives insights into the degree of equivariance to the three transformation types. Here the comparison between both models, the Classic baseline and the CNN-LEA, is of interest. For both models the presence of information regarding the translation is constant and neither shows a significant increase nor a significant decrease, indicating that it is simply preserved. This is explainable by looking at the windowing process used in a convolutional layer, a pooling layer, and a LEA. The result for every window will always end up in the same relative position. For both models, the information regarding the scale is classified more accurately in subsequent layers. This is also explainable by the windowing process but with respect to the data reduction in that the scale in the input layer has a larger effect on subsequent layers. For the orientation, the results on the test set show that a CNN like architecture loses the orientation information over subsequent layers. This does not prove invariance to rotation in CNN architectures but disproves the existence of equivariance to rotation. Interestingly, the CNN-LEA model not only preserves the orientation information but also generates better representations on subsequent layers. This seems to be an advantage of the LEA over a classical CNN architecture.

The results show that CNN architectures are not only invariant but also equivariant to translation and scale transformations. While the Classic baseline and CNN-LEA perform similar for changes in position and scale, the CNN-LEA has a clear advantage for orientation changes.

5.3 Intelligent Pooling

Pooling and the LEA perform a size reduction of the input images on a per-window basis. Pooling reduces every window per channel to a scalar value by a untrained reduction operation, e.g. max-pooling reduces to the maximum value. LEA is a trained operation that reduces every window, including all channels, to a vector

with a length according to the code length. Given these similarities, it is of interest how both methods directly compare. Since the LEA is a trained operation, it could at least fall back and learn to do something similar to pooling. That would suggest that a successfully trained LEA has the potential to be at least as good as pooling on the reduction task. The encoding based on the reconstruction objective could also show other advantages. This section, therefore, compares max-pooling with the LEA, specifically the classification performance of the POOL-FC baseline, the LEA-FC model with a VAE, and the LEA-FC model with a standard AE. Both AEs in this comparison use the fully-connected encoder and decoder variants.

From a theoretical standpoint, a LEA with a standard AE and fully-connected encoder should be able to learn and perform a max-pooling function. The universal approximation theorem states that an ANN can approximate any continuous function for a closed and bounded subset of $\mathbb{R}^n$ if it consists of a layer with non-linear activation functions and a subsequent layer with a linear activation function (Cybenko, 1989; Goodfellow *et al.*, 2017). Since the max function for two variables can be written as a linear combination of continuous functions defined for positive input values as shown in Eq. (5.1), the theorem applies.

$$\max(x, y) = \frac{1}{2}\left(x + y + |x - y|\right) \tag{5.1}$$

This is more obvious when rewriting the formulation using the ReLU activation function as shown in Eq. (5.2).

$$\max(x, y) = \frac{1}{2}\left(\phi_{\mathrm{ReLU}}\left(x + y\right) + \phi_{\mathrm{ReLU}}\left(x - y\right) + \phi_{\mathrm{ReLU}}\left(y - x\right)\right) \tag{5.2}$$

This formulation equals a fully-connected ANN with three neurons with ReLU activation function and a subsequent neuron with a linear activation function. Since the LEA with standard AE and fully-connected encoder in the test setup only has positive inputs in the range $[0, 1]$, a layer with ReLU activation function and a subsequent layer with linear activation function, it should be able to learn and perform a max-pooling function.

The test setup compares max-pooling against the LEA in terms of classification performance for different window sizes. This classification model for POOL-FC consists of the layer sequence *pool-fc-fc* and for LEA-FC of the layer sequence *lea-fc-fc*. To reiterate, the LEA is only trained based on the reconstruction objective and the fully-connected layers on the classification objective. Since the LEA in this test setup is directly connected to the input data, the LEA output is fully independent of the classification objective. The comparison is performed for the window sizes 2, 4, 8, 16, 32, and 64 on the dSprites dataset that consists of images with a size of 64×64×1 px. The stride is chosen in a way that no overlap or gap between windows occurs, i.e. the stride size is equal to the window size. Every

model and window size combination is fully trained 10 times over 100 epochs. This comparison is done for AEs with a code length of one and with a code length of 10. Since max-pooling reduces every window per channel to a scalar value and the input image has only one channel, the comparison against a LEA is only fair for codes of length one.

The results of this test for both code lengths can be seen in Fig. 5.5. As expected, a LEA with standard AE and fully-connected encoder performs at least as good as max-pooling. This does not conclude that the LEA performs max-pooling, in fact, the performance is consistently slightly better and therefore suggest that a better operation is learned. Interestingly, for small window sizes, i.e. a window size of 2×2 px, the restriction effect of the VAE seems to hurt the classification performance. This is the case for both code lengths. Other than that, all methods in the comparison with a code length of one perform relatively similar with the LEA based models slightly outperforming the max-pooling based model. For a code length of 10, the difference between max-pooling and the LEA is apparent. While max-pooling loses important information with every window size increase, the LEA mostly retains all information. The VAE based LEA even improves with larger window sizes. This is probably due to the restriction effect, in that for larger windows it is easier to find patterns that can be disentangled.

Since larger window sizes also denote larger stride sizes, these results indicate the possibility to use larger strides with larger window sizes. As a consequence of larger strides, a larger size reduction is achieved. This has the potential to reduce the number of layers within an ANN.

5.4 Sample Complexity

Artificial neural networks (ANNs) need lots of training data to learn a good function approximation. Improving the data efficiency, i.e. reducing the sample complexity, is an important goal. The LEA has properties that might have positive effects on the sample complexity. The AE is trained in a data-efficient way in that every input image is split into many windows and therefore increase the training sample size significantly. The fact that an AE learns to encode the input data, might be favorable as well. The number of training samples of individual classes depends on the number of occurrences within images instead of the number of images. For the reconstruction objective or more precisely the encoding, the network learns to focus on important features. On a local scale, this could help the network to focus on important parts of the input data and therefore a reduced number of necessary training samples. An additional factor could be the unsupervised learning of subclasses. This would add training information that was not explicit before. This adds features to the input data which could help to reduce the number of needed training samples.

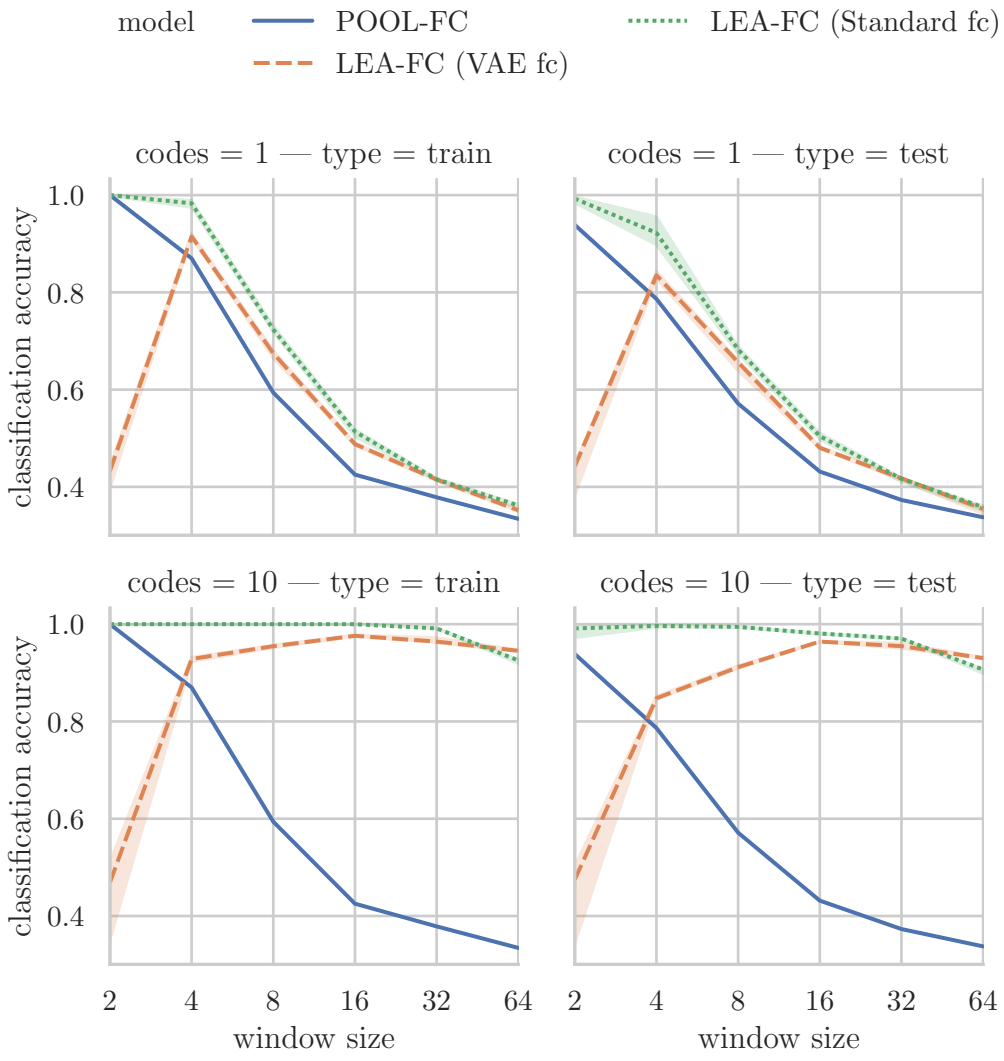

Figure 5.5: Classification accuracy on the dSprites dataset for different window sizes after 100 epochs training of three models for two code lengths of one and 10.

For the test of sample complexity, four subsets of the dSprite dataset are used. These four subsets consist of a selection of 10, 500, 5000, and 50 000 samples of the training set. Before the selection, the training set is shuffled to ensure a random distribution and therefore an even composition of target classes. No explicit measure is taken to ensure the class-balance. Since the dSprites dataset classification has three target classes, the smallest set of 10 samples could still contain three samples per class. Reducing the number of training samples still comes along with the risk of increasing the class-imbalance and therefore the risk for overconfidence. The shown classification loss results for training show the performance on the reduced training subset and for the test on the whole testing set.

The results of this test are shown in Fig. 5.6. Instead of the classification accuracy, the loss is displayed. This is due to the difference between both metrics in this case. For 10 and 500 training samples the loss of the Classic baseline increases while its accuracy remains constant. This is a clear indicator of overconfidence. While the Classic baseline is affected by overconfidence, the LEA is not. This highlights a potential advantage of the LEA operation. Also, the difference between training and testing results is way smaller. Hence, the LEA shows less overfitting compared to the Classic baseline. Important to note here that the accuracy for 10 and 500 training samples shows a bad classification performance for both models, only slightly above random guessing. The Classic baseline shows a faster convergence with both models converging to similar results when training long enough. Since in the LEA two objectives are optimized at the same time, this is an expected result.

5.5 Robustness Against Adversarial Attacks

This section evaluates the robustness of the LEA against adversarial attacks. For this evaluation, the CNN-LEA and Classic baseline models are trained for 100 epochs without significant overfitting on the dSprites dataset. These fully trained models are tested against several gradient-, score- and decision-based adversarial attacks. Since in most cases the result difference turned out to be very minor, as can be seen in Fig. A.1, the results of the CNN-LEA models are normalized around the results of the Classic baseline. These comparison results can be seen in Fig. 5.7. Note that the results for the VAE based models with Spatial Broadcast decoder are omitted since they failed to properly learn the classification objective. Including these would result in a distorted comparison due to different extrema.

Many of the gradient-based attacks, ADef, Deep Fool, and SLSQP to be specific, show a higher success rate than against the Classic baseline. At the same time, the ADef and SLSQP attacks need a slightly higher perturbation distance compared to the Classic baseline. These results still suggest that models incorporating LEAs are more vulnerable to gradient-based attacks. In contrast to this observation, the

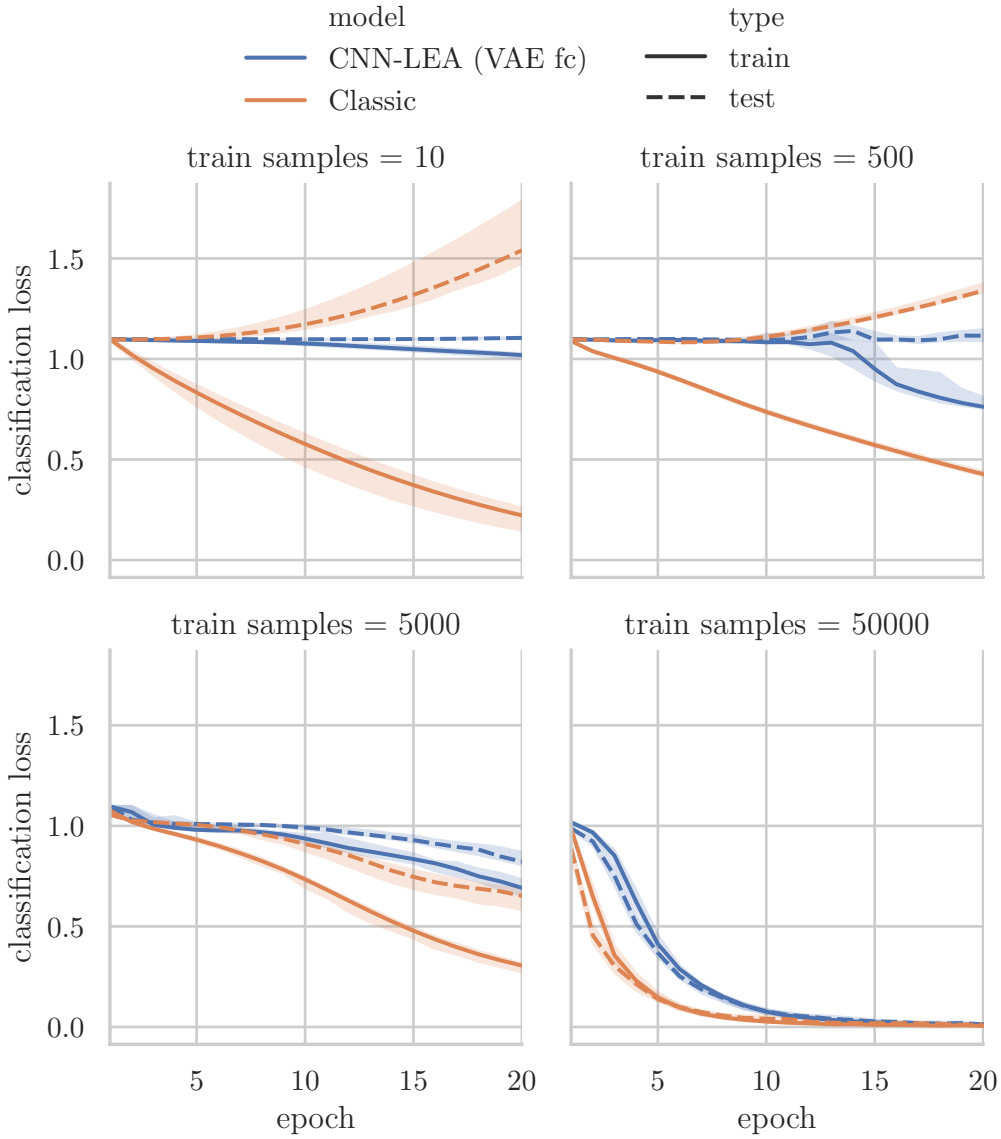

Figure 5.6: Classification loss per epoch for different sized training subsets of the dSprites dataset.

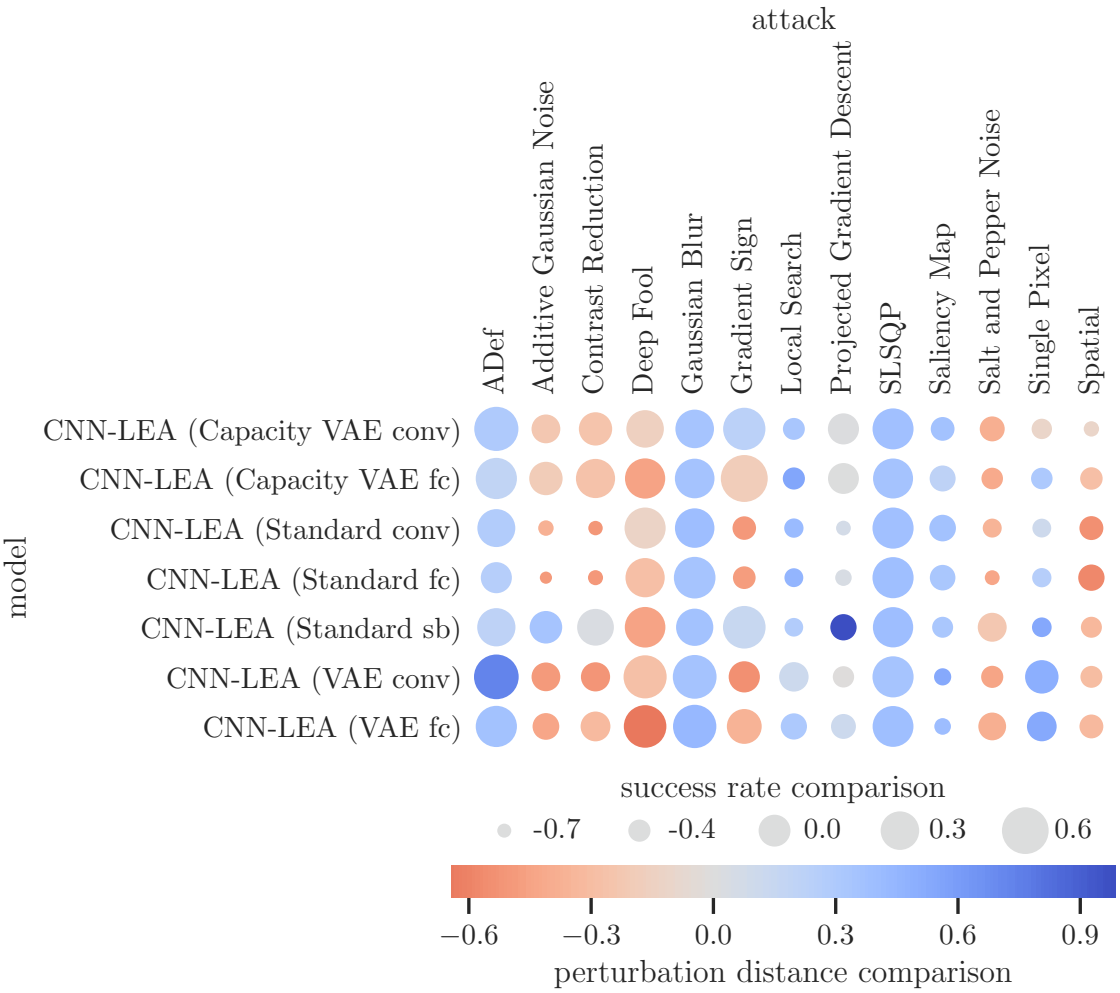

Figure 5.7: Comparison of robustness against adversarial attacks between CNN-LEA models with different AE setups and the Classic baseline. The zero values represent the Classic baseline. Smaller circles, i.e. negative values, show a lower success rate in comparison to the Classic baseline. Blue colors, i.e. positive values, show a larger perturbation distance in comparison to the Classic baseline. Evaluated on the dSprites dataset.

success rate of the Projected Gradient Descent attack is slightly lower and for the standard AE with Spatial Broadcast decoder the perturbation distance is slightly higher. This could be due to the restricted nature of the Projected Gradient Descent attack. Also, the robustness against the Saliency Map attack is slightly better in terms of success rate and perturbation distance.

The robustness against score-based attacks shows the only major differences. Especially against the standard AE and capacity-controlled VAE models, the success rate of the Single Pixel attack is significantly lower. An argument could be that this is based on the dimensionality reduction nature of the encoding process but the missing improvement for the VAE models contradicts this. It is also still an open question why the VAE and the capacity-controlled VAE show different results.

The robustness against decision-based attacks completely depends on the specific attack. Only the Gaussian Blur attack shows an improvement in terms of a higher perturbation distance. The Salt and Pepper Noise attack is the only attack that shows a lower success rate. Since the Spatial attack performs spatial transformations, it is of particular interest. The attack performs spatial transformations and the dSprites dataset contains . Since the generative factors of the dSprites dataset are spatial transformations, it would have been expected to see major improvements against this type of attack for this dataset. While the success rate is overall lower, it did not meet those expectations for the perturbation distance.

The differences between the results for different models seems to be more nuanced. This evaluation did not reveal a clear pattern or obvious advantages. Important to note is that the above observations are based on minor or even infinitesimally small differences. Even though the results are averages of 10 different runs, as a consequence of the normalization the differences could also be the result of low fluctuations. However, as mentioned in the beginning, except for the score-based attacks the results did not show any significant improvement or deterioration in robustness against adversarial attacks.

5.6 Regularization Effect

The AE objective can also be seen as auxiliary regularization for the classification objective. The Decoder-free baseline uses only the encoder-pathway and ignores the decoder as well as the AE objective. Since the encoder-pathway is trained using the classification objective, it is just a simple ANN. More specifically, if the fully-connected encoder-decoder variant is used, the LEA only acts as a fully-connected layer with connections based on the windowing configuration. The question is, how adding the AE objective would affect the classification objective.

The test setup for this analysis uses the LEA-only model. This is a LEA plus fully-connected layers trained jointly on the classification and AE objective. This joint training is performed by defining a joint loss. The joint loss is simply a

weighted combination of all losses. This combination is shown in Eq. (5.3),

$$L_{joint} = \tau \cdot L_{class} + \lambda \cdot \sum L_{lea} \tag{5.3}$$

where L_{class} is the classification loss, L_{lea} is the loss of a LEA, and L_{joint} is the joint loss. If multiple LEAs are used, their joint loss is simply the sum without any weighting. The scalar weights τ and λ weight the classification and total AE loss, respectively. The weight λ of the total AE loss is referred to as LEA weight.

To adjust the balance between both losses, it suffices to adjust only one of both weights. In this test setup, the classification weight τ is set to one, and only the LEA weight λ is adjusted. If the LEA weight is set to zero, the AE loss does not affect the training and therefore the LEA-only model reduces to the Decoder-free baseline. This test is intended for to analysis of the effect of adding the AE objective to the classification objective, i.e. adding the AE loss. By adjusting the LEA weight, it is possible to observe the respective effect on classification and reconstruction performance. In this test setup, the capacity-controlled VAE with the fully-connected encoder-decoder variant is used. The test is performed on the CIFAR-10 dataset with a window size of 8×8 px and a code length of 10. This evaluation is done for a fixed set of LEA weights, namely the values 1×10^2, 1×10^1, 1, 1×10^{-1}, 1×10^{-2}, 1×10^{-3}, and 1×10^{-4}. For every weight configuration, 10 runs are fully trained over 200 epochs.

The results of this test are shown in Fig. 5.8. This plot compares the effect of different LEA weights on classification and reconstruction loss. The reported loss values are based on the epoch with the lowest test classification loss, namely epoch 19. The expected result is that for a higher LEA weight the reconstruction loss is lower. This confirms that in a joint training the AE objective helps to encourage a reconstructible output. Therefore the AE objective strengthens the preservation of information about the input data. Contrary to the reconstruction loss, the classification loss is lower for a lower LEA weight. Since the AE objective operates as a regularization to the classification objective, it is expected that it has an impact on the classification performance. The results suggest that the classification performance generally is better without the auxiliary AE objective. This result is unexpected since this auxiliary objective is supposed to support the classification objective.

This evaluation does not show a positive correlation between the AE objective and the classification objective. With the right balance, both objectives can be pursued without significantly harming the other. It can be observed that increasing the LEA weight further than 1×10^{-1} does not result in any further improvement in reconstruction quality. The reverse seems true for the classification performance, decreasing the LEA weight further than 1×10^{-1} does not lead to a significant improvement in test loss. Therefore, a LEA weight of 1×10^{-1} is the sweet spot in which both objectives are well optimized. Interestingly, the training loss of

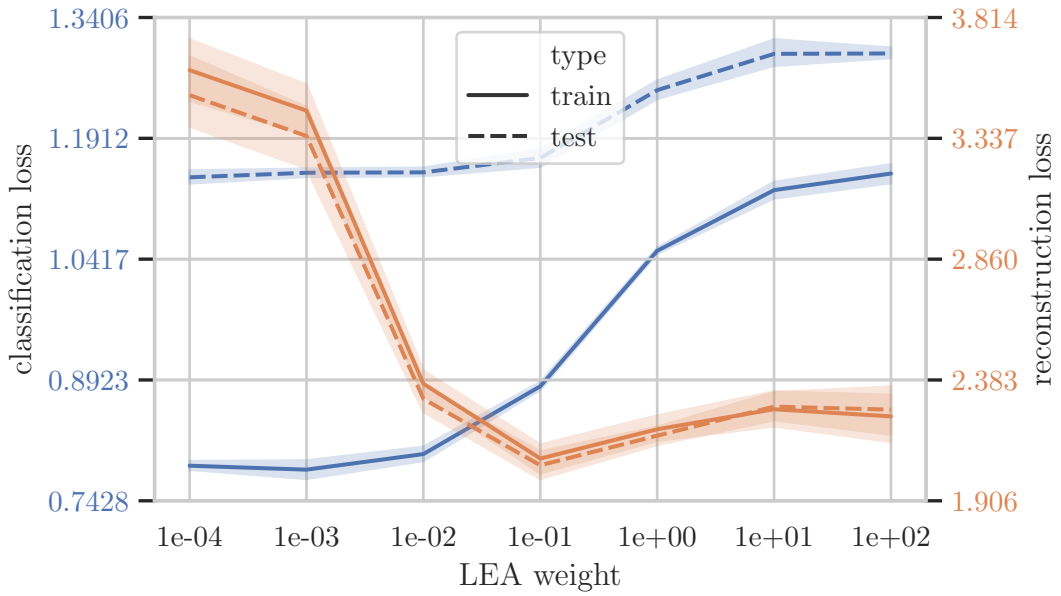

Figure 5.8: Observation of the correlation between the classification and AE objective in terms of classification and reconstruction loss on the CIFAR-10 dataset. While the classification loss (blue) increases for higher LEA weights, the reconstruction loss (orange) decreases.

the classification is more affected by the change in LEA weight. It seems that a higher LEA weight leads to lower overfitting. This is also true for a LEA weight of 1×10^{-1}. A LEA weight, therefore, encourages the preservation of information and improves the overfitting behavior while not hurting the classification performance.

Chapter 6

Discussion

The implementation developed for the evaluation of the LEA records many metrics. A clear and flexible implementation was the priority to make continuous work, experiments, and analysis easier. Both, the recording of metrics and the flexible implementation, are at the expense of training time. Compared to CNN like baselines, the training took up to 10 times as long. This is especially true for larger image inputs and larger window sizes. Without optimization to runtime performance, this makes experiments with larger datasets like ImageNet difficult. Another point related to this issue is the performance comparison of this operation. The results of this work compare performance by epochs. This is reasonable in that it compares models on the base of training repetitions. It makes the comparison of models independent to hardware and software related variations. However, inference time matters in real-world applications where a model with faster computation time might be preferred. This was not the focus of this work and computation times neither for the training nor for inference are reported. Nonetheless, there is a lot of optimization potential that could decrease the computation time of this implementation. Especially for inference, the LEA can have a significant speed-up by removing the decoder path that is only needed during training.

As mentioned before, the reported performance on the used datasets in terms of accuracy and loss is nowhere near the state-of-the-art. In light of this work that is not an issue. The models are simple and no additional tricks, which are common in state-of-the-art models, are used. This helps to avoid side effects and therefore to ensure that reported results are due to the LEA and not based on other already proven techniques. This also helps to reduce the training time and makes research more efficient. The shown results are also valid for smaller more simpler models and do not rely on large state-of-the-art architectures. However, this leaves open how the LEA performs compared to or with state-of-the-art models and for that matter if it is competitive at all.

On a similar note, another shortcoming is that this work evaluated the LEA only on toy datasets. To evaluate ML models, datasets like MNIST, Fashion-MNIST, CIFAR-10, and dSprites are commonly used based on their ease of use. It could be argued that these datasets are too simple and do not reflect current real-world problems, i.e. implying that an ML model that worked well on these datasets fails

when applied to real-world problems. However, analyzing the effects of the LEA on toy datasets is not an issue. What is an issue, is that for the analysis of the encoding and especially the disentangling process these datasets might not be the perfect fit. These processes start to get interesting when the encoding is done for larger windows containing whole objects. However, the image size of the used datasets is too small to allow larger windows and these images do not contain larger subobjects. To research the capabilities of local unsupervised learning it could help to use a dataset with images that contain a fixed set of subobjects which appear repeatedly within the same images.

Optimizing for a better disentanglement within the code turned out to be very difficult. The existing metrics for disentanglement, the BetaVAE metric (Higgins *et al.*, 2017), the Separated Attribute Predictability score (Kumar *et al.*, 2017), the Mutual Information Gap (Chen *et al.*, 2018), the DCI Disentanglement (Eastwood and Williams, 2018), and the FactorVAE metric (Kim and Mnih, 2018), use the dataset's ground truth generative factors to measure disentanglement. Simply put, the expectation is that the data is disentangled into the generative factors and if these are known, the degree of disentanglement can be measured. In other words, these disentanglement metrics only work supervised. Even though the generative factors of the dSprites dataset are known, these are not the required factors to measure the degree of disentanglement with respect to the LEA. These generative factors are only the explanatory factors of the highest hierarchy but local explanatory factors are unknown. Given any number of hierarchical levels, this is true for all datasets. Locatello *et al.* (2018) state that there is currently no unsupervised disentanglement metric that helps to optimize hyperparameters and that supervision is necessary. The code maps of the LEA, are theoretically ideal to detect disentanglement via supervision in the form of manual interpretation. First of all, images are one of the best interpretable data formats, second of all every code map should represent an explanatory factor in a well-disentangled representation. While this seems to be true, it is not clear how different disentangled code maps compare. Given two relatively similar code maps or both show different but seemingly meaningful (interpretable) results, then there seems to be no clear method to assess one of both code maps to be better disentangled. An unsupervised disentanglement metric that is independent of the ground truth explanatory factors is needed.

The empirical results of this work could neither prove nor disprove the usefulness of disentanglement for supervised tasks. Bengio *et al.* (2013) suggest that disentangled representations lead to a decreased sample complexity for supervised learning tasks. Contrary to this, Locatello *et al.* (2018) could not find evidence that better disentanglement measured by the above-mentioned disentanglement metrics leads to a decreased sample complexity. van Steenkiste *et al.* (2019) countered that by providing empirical results on what basis they claim compelling evidence that better disentanglement leads to a decreased sample complexity. While this work can report a better overfitting behavior for models trained with fewer data and dis-

entangled representations and also a decreased sample complexity, the respective supervised learning performance was not sufficient. These results could get more conclusive in the future when better disentanglement methods are developed.

Chapter 7

Conclusion and Future Work

In direct comparison with max-pooling, the LEA does learn a slightly better reduction function in terms of forwarding task-relevant information. For larger code lengths the LEA outperforms max-pooling, except for small window sizes. The usage of larger windows allows for a greater dimensionality reduction. This work provides interesting insights into how CNNs handle transformations. The empirical results show that CNNs are equivariant and not invariant to spatial transformations like translation and scale. Only the very last layer drops all spatial information and reduces the information to the expected output results. Evidence for a decrease in sample complexity for a supervised task could be found. The LEA did not show significant improvements for the robustness against adversarial attacks.

This work confirms that learning to discriminate explanatory factors through unsupervised disentanglement is possible. While this work could not show the learning of a hierarchical structure of subobjects, the learned code maps did show coherent patterns and therefore some kind of feature encoding. The results prove that the disentanglement of local representations can be pursued without worsening the respective supervised performance. However, the empirical results of this work show that stronger restrictions on the local representations with respect to the reconstruction and disentanglement objective do not correlate with a better classification performance. Nonetheless, unsupervised disentanglement is still a challenging and unsolved problem. The LEA could greatly benefit from stronger disentanglement techniques. Considering the current state-of-the-art, it seems that disentanglement is still not understood very well and that it mostly depends on human interpretation. Research should be made to properly define disentanglement in a way that it is measurable without the need for supervision. This would lay the foundation for future research on techniques competing and improving the state-of-the-art. Subsequently, further research is necessary to prove that interpretable explanatory factors of hierarchical subobjects are useful and learnable through the proposed LEA operation. Further research should restrict the LEA as much as possible for an analysis independent of other side effects. This research should answer the question if explanatory factors for intermediate representations improve the discrimination of higher-order objects. To answer that question a special toy dataset containing labels for every level of hierarchies that has a challenging ob-

jective while having a minimum of side effects should be generated. It is an open question how this dataset and the corresponding objective should look like.

The LEA shows an improved equivariance to spatial transformations that is probably due to the pixelwise reconstruction objective. The observed equivariance to translation and scale in CNNs could solely be based on the windowing function. In other words, it is unclear if equivariance to spatial transformations is useful for the joint supervised task. A hierarchical structure of objects could also be represented by neighborhood relations invariant to the global spatial transformations. Especially the disentanglement could improve since it would not need to forward spatial information. In future work, it would be highly interesting to investigate the effect of using a semantic-based instead of a pixelwise reconstruction loss for the AE training. Techniques like the Siamese network (Bromley *et al.*, 1994) or the VAEGAN (Larsen *et al.*, 2016) could be used as a similarity metric. The semantic reconstruction objective could be supported by using spatial transformers (Jaderberg *et al.*, 2015) for the training of the similarity metric. This could be extended to a split of the code into semantic and spatial information. The semantic pathway could be used as a shortcut connection to the spatial pathway for the discarding of semantic information.

Appendix A

Supplementary Material

A.1 Training Configurations

The graphics and plots shown in this work resulted from the following training configurations. For gradient optimization, the Adam optimizer with a learning rate of 1×10^{-4} did show the most stable results and therefore was used for all experiments. The convolutional layers and LEAs were used with zero-padding so that no input values are dropped. All layers were used with the ReLU activation function to allow unrestricted positive values. If not specified otherwise, datasets were unlimited, meaning that the training was performed on the whole dataset. The batch size was 256 for MNIST, 128 for Fashion-MNIST and CIFAR-10, and 64 for dSprites. If hyperparameters are not specified in the individual configurations listed below, they are set to the default values as defined by TensorFlow. If other details are not mentioned below, the implementation details as explained in this work applies.

Figs. 3.3 to 3.5 of the AE evaluation are based on an AE training over 800 epochs on the Fashion-MNIST dataset and with a code length of 10. All AE types are used with default parameters for the standard AE and the VAE, with $a = 1.3$ and the top-5 activations for the k-sparse AE, and a scale of 5 and a capacity control value of 20 with a linear increase for the capacity-controlled VAE. All encoder-decoder variants are used with default parameters for the fully-connected and 32 kernels for the convolutional and spatial broadcast variants.

Figs. 4.1 and 4.2 show examples of the reconstruction and classification metrics that are based on an alternate CNN-LEA training over 20 epochs on the dSprites dataset. As AE type the VAE with a fully-connected encoder-decoder and a code length of 5 is used. The LEA uses a window size of 4×4 px and a stride of 2×2 px.

Fig. 4.3 shows examples for the confidence visualization are based on an alternate CNN-LEA training over 100 epochs on three datasets. The used AEs use a fully-connected encoder-decoder with a code length of 10. The LEA uses a window size of 4×4 px and a stride of 2×2 px. While for the Fashion-MNIST dataset a standard AE is used, for the CIFAR-10 and MNIST dataset a capacity-controlled VAE with a scale of 10 and a capacity control value of 25 with a linear increase is used.

Fig. 5.1 shows examples for adjustments of the code of an AE trained over 600

epochs on the dSprites dataset. The used AE is a capacity-controlled VAE with a scale of 10, a capacity control value of 25, an increase over 100 epochs and a code length of 10. The encoder-decoder is the spatial broadcast with 64 kernels.

Fig. 5.2 shows the 10 code maps of a LEA with a window size of 4×4 px and a stride of 1×1 px. It uses a fully-connected encoder-decoder. For the CIFAR-10 dataset, a capacity-controlled VAE with a scale of 10, a capacity control value of 25 is trained over 200 epochs with an increase over the first 25 epochs. For the dSprites, Fashion-MNIST, and MNIST dataset a capacity-controlled VAE with a scale of 10, a capacity control value of 25 is trained over 800 epochs with a linear increase.

Fig. 5.4 shows the loss in spatial information over multiple layers by using the known generative factors of the dSprites dataset. The network under test is a CNN-LEA using a VAE with fully-connected encoder-decoder, a code length of 10, a window size of 4×4 px, and a stride of 2×2 px. Alternate training of the CNN-LEA is done for 50 epochs. The spatial classifier is trained for 20 epochs.

Fig. 5.5 shows the comparison against max-pooling in a per window size comparison. Alternate training of both methods on the dSprites dataset is done for 100 epochs. The two AEs, the VAE, and the standard AE are evaluated using a fully-connected encoder-decoder. The comparison is performed for a code length of one and a code length of 10. The window sizes 2, 4, 8, 16, 32, and 64 are evaluated. For all window size results, the stride was equal to the respective window size.

Fig. 5.6 shows the test on the sample complexity. Alternate training of the CNN-LEA is done for 20 epochs on the dSprites dataset. A VAE with a fully-connected encoder-decoder, a window size of 4×4 px, a stride of 2×2 px, and a code length of 5 is used. In independent runs, the dSprites dataset is limited to 10, 500, 5000, and 50 000 samples.

Figs. 4.5, 5.7 and A.1 show heatmaps for the comparison of robustness against adversarial attacks. The CNN-LEA configurations were evaluated on their robustness after an alternate training on the dSprites dataset for 100 epochs. The LEA used a window size of 4×4 px, a stride of 2×2 px, and a code length of 10. As AEs the standard AE, the VAE, and the capacity-controlled VAE with a scale of 10, a capacity control value of 25, and a linear increase are used. All encoder-decoder variants are evaluated.

Fig. 5.8 shows the effect of adding the AE objective as an auxiliary objective to a supervised task. This is evaluated by jointly training an AE for 200 epochs on the CIFAR-10 dataset. A capacity-controlled VAE with a scale of 10, a capacity control value of 25, an increase until epoch 25, a fully-connected encoder-decoder, a window size of 8×8 px, a stride of 4×4 px, and a code length of 10 is used. The LEA weight is set to 1×10^2, 1×10^1, 1, 1×10^{-1}, 1×10^{-2}, 1×10^{-3}, and 1×10^{-4}. Only the results of epoch 19 are shown.

A.2 Inconclusive Robustness Results

The difference in perturbation distance between all models for every attack is minor. Visualizing the existing differences, as in Fig. A.1 is made worse by different attacks resulting in different scales for the perturbation distance. Comparing all models and all attacks with each other with a color range for the perturbation distance simply highlights the attack with the highest perturbation distances. Minor differences in perturbation distance are lost. Only the success rate shows major changes for the Single Pixel attack.

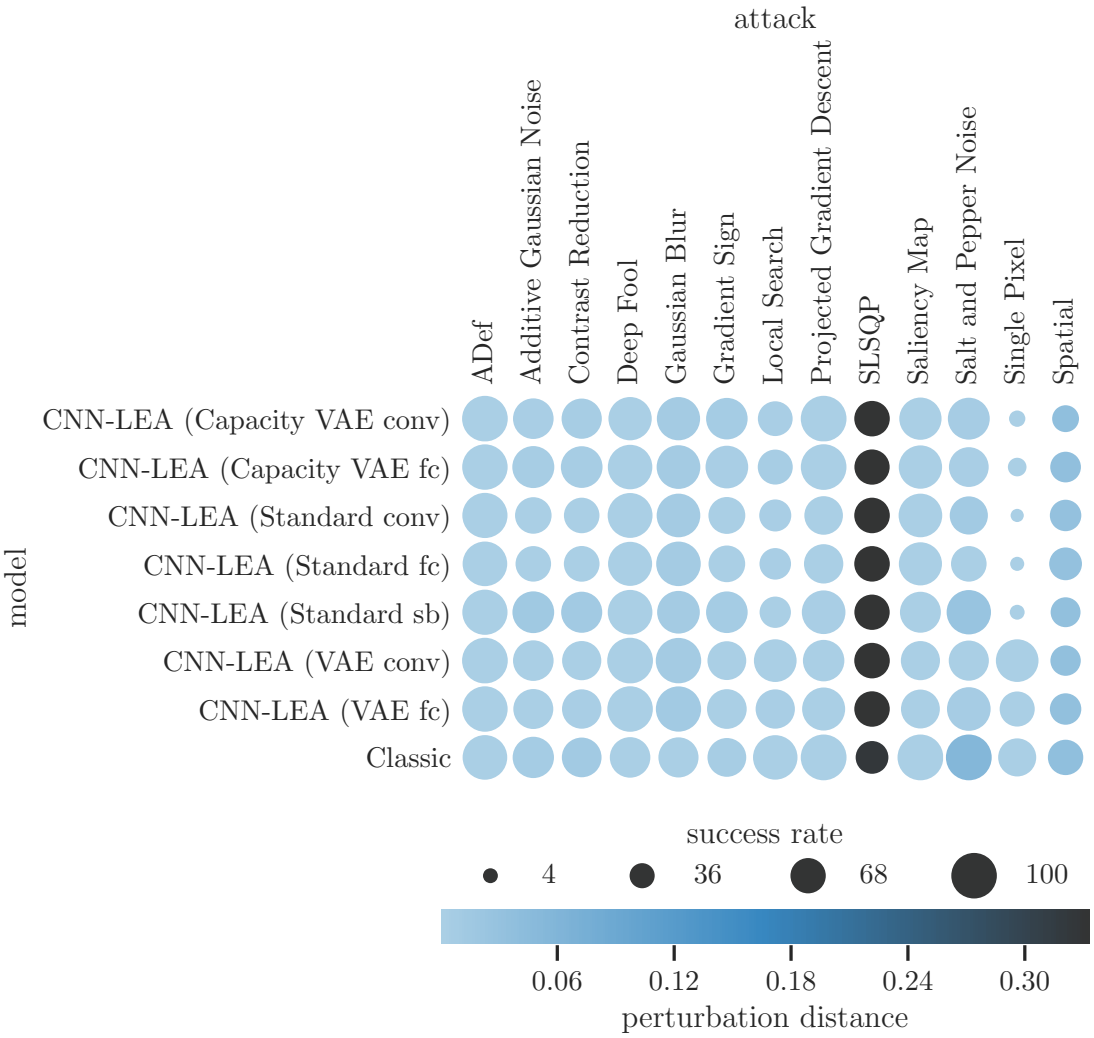

Figure A.1: A heatmap of the robustness against adversarial attacks with colorization by perturbation distance and circle size by success rate. All minor differences are mostly lost. Evaluated on the dSprites dataset. Smaller and darker circles are better.

Abbreviations

Adam adaptive moment estimation

AE autoencoder

ANN artificial neural network

BagNet deep bag-of-features model

CNN convolutional neural network

CPU central processing unit

ELU exponential linear unit

GPU graphics processing unit

KL Kullback–Leibler

LEA locally embedded autoencoder

MAE mean absolute error

ML machine learning

MLP multilayer perceptron

MSE mean squared error

MTL multi-task learning

RAM random-access memory

ReLU rectified linear unit

SGD stochastic gradient descent

UI user interface

VAE variational autoencoder

Bibliography

Alaifari, R., Alberti, G. S., and Gauksson, T. (2018). Adef: an iterative algorithm to construct adversarial deformations. *International Conference on Learning Representations*.

Ballard, D. H. (1987). Modular learning in neural networks. *Conference on Artificial Intelligence*, pages 279–284.

Baxter, J. (1995). Learning internal representations. *Computational Learning Theory*, pages 311–320.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(8), 1798–1828.

Brendel, W. and Bethge, M. (2019). Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet. *International Conference on Learning Representations*.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a "siamese" time delay neural network. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 07 of *6*, pages 737–744. Morgan-Kaufmann.

Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. (2018). Understanding disentangling in ß-VAE. *arXiv*.

Cauchy, A.-L. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comptes Rendus Hebd. Seances Acad. Sci.*, **25**, 536–538.

Chapelle, O., Schölkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. MIT Press.

Chen, T. Q., Li, X., Grosse, R. B., and Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 31, pages 2610–2620. Curran Associates, Inc.

Clearwater, S. H., Cheng, T.-P., Hirsh, H., and Buchanan, B. G. (1989). Incremental batch learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 366–370. Elsevier.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUs). *International Conference on Learning Representations*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, **2**(4), 303–314.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv*, page 31.

Eastwood, C. and Williams, C. K. I. (2018). A framework for the quantitative evaluation of disentangled representations. *International Conference on Learning Representations*.

Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., and Madry, A. (2017). A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv*.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36**(4), 193–202.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. *International Conference on Artificial Intelligence and Statistics*, **15**, 315–323.

Goodfellow, I., Bengio, Y., and Courville, A. (2017). *Deep Learning*. The MIT Press.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *International Conference on Learning Representations*.

Hadamard, J. (1908). *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*. Mémoires présentés par divers savants à l'Académie des sciences de l'Institut de France: Éxtrait. Imprimerie nationale.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). ß-VAE: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations*, **3**.

Higgins, I., Amos, D., Pfau, D., Racaniere, S., Matthey, L., Rezende, D., and Lerchner, A. (2018). Towards a definition of disentangled representations. *arXiv*.

Hinton, G. E. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507.

Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In T. Honkela, W. Duch, M. Girolami, and S. Kaski, editors, *Artificial Neural Networks and Machine Learning*, pages 44–51. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 28, pages 2017–2025. Curran Associates, Inc.

Kang, Z., Grauman, K., and Sha, F. (2011). Learning with whom to share in multi-task feature learning. *International Conference on Machine Learning*, **2**(3), 4.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *ARS Journal*, **30**(10), 947–954.

Kim, H. and Mnih, A. (2018). Disentangling by factorising. *International Conference on Machine Learning*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kosiorek, A. R., Sabour, S., Teh, Y. W., and Hinton, G. E. (2019). Stacked capsule autoencoders. *arXiv*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Kumar, A., Sattigeri, P., and Balakrishnan, A. (2017). Variational inference of disentangled latent concepts from unlabeled observations. *International Conference on Learning Representations*.

Kuzminykh, D., Polykovskiy, D., and Zhebrak, A. (2018). Extracting invariant features from images using an equivariant autoencoder. *Asian Conference on Machine Learning*, **95**, 438–453.

Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. *International Conference on Machine Learning*, **48**(33), 1558–1566.

le Rond d'Alembert, J. (1754). *Recherches sur différents points importants du système du monde*. David l'aine, Paris.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1**(4), 541–551.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg.

Lenc, K. and Vedaldi, A. (2014). Understanding image representations by measuring their equivariance and equivalence. *IEEE Conference on Computer Vision and Pattern Recognition*.

Linnainmaa, S. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. mathesis, University of Helsinki.

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, **16**(2), 146–160.

Locatello, F., Bauer, S., Lucic, M., Rätsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2018). Challenging common assumptions in the unsupervised learning of disentangled representations. *International Conference on Machine Learning*.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning*.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*.

Makhzani, A. and Frey, B. (2013). k-sparse autoencoders. *International Conference on Learning Representations*.

Mathieu, E., Rainforth, T., Siddharth, N., and Teh, Y. W. (2019). Disentangling disentanglement in variational autoencoders. *International Conference on Machine Learning*, **97**(36), 4402–4412.

Matthey, L., Higgins, I., Hassabis, D., and Lerchner, A. (2017). dsprites: Disentanglement testing sprites dataset. GitHub Repository.

Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2015). Deepfool: a simple and accurate method to fool deep neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*.

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. (2016). Universal adversarial perturbations. *IEEE Conference on Computer Vision and Pattern Recognition*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *International Conference on Machine Learning*, pages 807–814.

Narodytska, N. and Kasiviswanathan, S. P. (2016). Simple black-box adversarial perturbations for deep networks. *arXiv*.

Nixon, M. and Aguado, A. S. (2013). *Feature Extraction and Image Processing*. Newnes, 3rd edition.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016). The limitations of deep learning in adversarial settings. *IEEE European Symposium on Security and Privacy*.

Poole, D. L. and Mackworth, A. K. (2017). *Artificial Intelligence*. Cambridge University Press, New York, 2nd edition.

Ranzato, M. A. and Szummer, M. (2008). Semi-supervised learning of compact document representations with deep networks. *International Conference on Machine Learning*.

Rasmus, A., Valpola, H., Honkala, M., Berglund, M., and Raiko, T. (2015). Semi-supervised learning with ladder networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 28, pages 3546–3554. Curran Associates, Inc.

Rauber, J., Brendel, W., and Bethge, M. (2017). Foolbox: A python toolbox to benchmark the robustness of machine learning models. Technical report, Bethge Lab.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, **22**(3), 400–407.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, **115**(3), 211–252.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning*. Cambridge University Press.

Shen, J. (2017). *Transformation Invariance and Equivariance in Deep Learning*. phdthesis, University of Chicago, Chicago, Illinois.

Shepard, R. N. and Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science*, **171**(3972), 701–703.

Siddharth, N., Paige, B., van de Meent, J.-W., Desmaison, A., Goodman, N. D., Kohli, P., Wood, F., and Torr, P. H. S. (2017). Learning disentangled representations with semi-supervised deep generative models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 30, pages 5925–5935. Curran Associates, Inc.

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). Ladder variational autoencoders. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 29, pages 3738–3746. Curran Associates, Inc.

Sobel, I. and Feldman, G. (1968). An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *International Conference on Learning Representations*, page 10.

Thom, M. and Palm, G. (2013). Sparse activity and sparse connectivity in supervised learning. *Journal of Machine Learning Research*, **14**(1), 1091–1143.

Valpola, H. (2015). From neural PCA to deep unsupervised learning. In E. Bingham, S. Kaski, J. Laaksonen, and J. Lampinen, editors, *Advances in Independent Component Analysis and Learning Machines*, chapter 8, pages 143–171. Academic Press.

van Steenkiste, S., Locatello, F., Schmidhuber, J., and Bachem, O. (2019). Are disentangled representations helpful for abstract visual reasoning? *arXiv*.

Ward, J. (2015). *The Student's Guide to Cognitive Neuroscience*. Taylor & Francis Ltd., 3rd edition.

Watters, N., Matthey, L., Burgess, C. P., and Lerchner, A. (2019). Spatial broadcast decoder: A simple architecture for learning disentangled representations in VAEs. *International Conference on Learning Representations*.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. phdthesis, Harvard University, Cambridge, Massachusetts.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv*.

Yuan, X., He, P., Zhu, Q., and Li, X. (2017). Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–20.

Zhang, Y., Lee, K., and Lee, H. (2016). Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. *International Conference on Machine Learning*.

Zhou and Chellappa (1988). Computation of optical flow using a neural network. *IEEE International Conference on Neural Networks*.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum	Unterschrift