
Goethe University Frankfurt
Faculty 12
Department of Computer Science

# Effects of Incrementally Increasing Latent Dimensionality in Autoencoders

**David Vogenauer**

Submitted in partial fulfillment
of the requirements for the degree of
**Bachelor of Science in Computer Science**

Supervisor: Prof. Dr. Matthias Kaschube
Second Examiner: Prof. Dr. Jochen Triesch

April 22, 2025

I am deeply grateful to Deyue, Jonas, and Matthias, as well as the entire FIAS research group, for their warm welcome, unwavering support, and for generously sharing their vast knowledge.

**Abstract**

The Developing Autoencoder (Dev-AE) is a variant of the standard autoencoder that incrementally adds neurons to its latent space during training, loosely inspired by the increase in dimensionality in neural cortices during development. Comparison of the Dev-AE to conventional autoencoder (AE) models aims to provide an understanding of the effects of the developing training method. The dynamic growth reveals significant findings: neurons introduced at different stages learn disentangled features and capture distinct frequencies; there is a hierarchy of importance, where reconstruction is driven by earlier neurons, while later neurons largely influence classification; and the sparsity observed in the latent space extends to the hidden layers. These results demonstrate that the small, biologically motivated change leads to disentangled, sparse representations, greatly boosting model interpretability.

# Contents

# 1 Introduction

Artificial neural networks are often considered to be black boxes, and the task of understanding what they learn during training is important for further improvements [1]. One promising route to finding more interpretable models is to draw on biological foundations. This is the approach of the Developing Autoencoder (Dev-AE) model introduced in [2], which is inspired by the increasing dimensionality of the brain during early development, leading to a coarse-to-fine learning approach. A prime example of this behavior is found in the visual cortex. Infants first recognize broad, dominant features before refining their perception to capture fine-grained details [3].

The Dev-AE model modifies the conventional autoencoder (AE) structure to represent an increase in dimensionality. Autoencoders reduce their input information to a latent space, capturing only the most dominant features. The Dev-AE increases the dimensionality of this latent space by incrementally increasing its size during training. The previous work proved that the Dev-AE achieves functionally equivalent performance to conventional AE models but leaves room for future research into understanding the differently structured latent spaces. This thesis adapts the approach presented in [2] and aims to answer the following question: How does an incremental increase in latent dimensionality affect representation learning?

To answer this, Dev-AEs are compared to conventional AEs initialized with equivalent hidden layers and a fixed latent space. These models are implemented multilayer perceptrons (MLP) autoencoders trained on the MNIST dataset, and convolutional neural network (CNN) autoencoders trained on CIFAR-10. Using two different datasets and architectures aims to provide general findings across various network types using the increasing dimensionality approach. The effects are measured by analyzing feature disentanglement in the latent space, importance of bottleneck neurons for reconstruction versus classification, and sparsity of activation, measured in both the latent space and other hidden layers. The results show that the biologically motivated developing training manner contributes to the beneficial properties that are inherent in the dynamic training method, without the need for manual tuning.

The remainder of this thesis is structured as follows. First, a foundation for both biological neural development and computational models is provided, supported by relevant literature. This is followed by the methodology used to implement and analyze the models. Then the core findings are presented, revolving around three main aspects: disentanglement, hierarchical ordering, and sparsity. Finally, the results are discussed, highlighting the learnings made about the Dev-AE and implications for practical applications.

# 2 Background and Literature Review

## 2.1 Biological Foundations of Neural Development

The nervous system perceives a very large amount of information. To make sense of these large amounts of high-dimensional data, they must first be reduced to their essential features [4]. This process is known as feature extraction or dimensionality reduction. The result is a lower-dimensional, compressed space, the latent space, whose dimensionality is aptly named latent or intrinsic dimensionality. In early development, the brain drastically lowers the complexity of external stimuli, keeping only broad features and discarding the finer, but potentially important details. Over time, the dimensionality gradually increases, allowing for a heavily compressed, yet well-rounded representation of relevant features [3]. Such encoding is achieved by ignoring irrelevant input and filtering out redundant patterns [4].

The increase in dimensionality accompanies a decrease in local correlation. Local correlation is in part responsible for the comparable response of neurons in close proximity to the same stimuli. The decrease in local correlation is linked to specialization [5]. Neurons begin to

fire independently of their neighbors and operate more freely: the ability to handle more complex structures emerges. For example, vision initially recognizes broad features and eventually progresses to distinguishing finer details [3]. Similarly, the auditory system evolves to discern intricate sounds from loud background noise [6]. This tendency is not limited to the visual and auditory cortex. Using in-vivo calcium imaging, spontaneous activity was measured across different cortical regions in ferrets [5]. All tested regions in the animals proceeded to show an increase of dimensionality (see Figure 1).

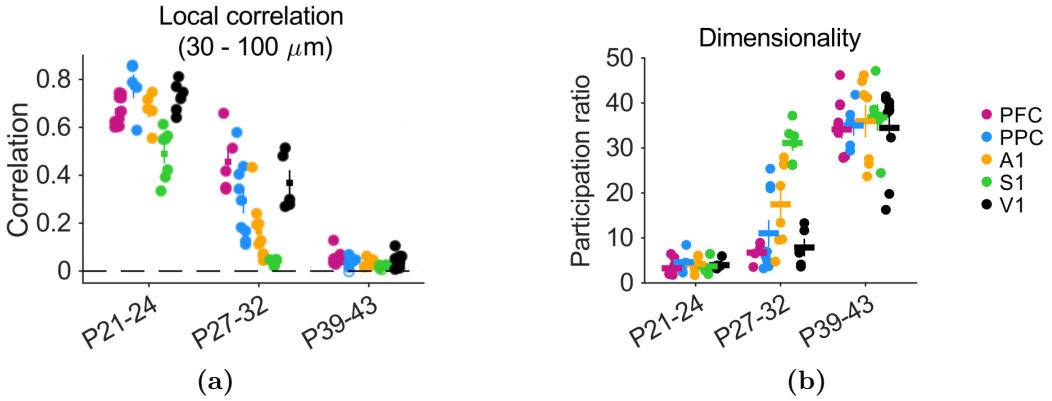



**Figure 1:** Developmental changes in spontaneous activity patterns across cortical regions in young ferrets. (a) from [5] and (b) from [7]. During maturation, dimensionality increases as local correlation decreases in five cortical areas: prefrontal cortex (PFC), posterior parietal cortex (PPC), primary auditory (A1), somatosensory (S1), and visual (V1) cortices.

Different regions of the cortex process different types of data. However, due to the neurons' further specialization within a region, external stimuli may only elicit a reaction from a small number of neurons. The stimulus that prompts a maximal response from a particular neuron is called its receptive field, a combination of two factors: the feature represented by the stimulus and its location in the sensory field [8]. For instance, neurons of the primary visual cortex will activate in response to features like edges, orientation, and motion, and they also activate if the input hits their specific area of the retina [9]. Each neuron thereby specializes in specific parts of the overall visual input.

In order to describe this specialization, where neurons respond to distinct features instead of a mixture of attributes, the term disentanglement is borrowed from machine learning [10]. While the brain does not separate properties in the same way a digital neuron does, parallels can be drawn. As an illustration, the brain's ability to recognize an object under different circumstances is similar to disentanglement. Despite changes in the environment, the object's core features remain stable, as they are independent of those variations. Its representation is abstracted and high-level features, such as shape, are maintained [11]. In this regard, disentanglement can help decrease the influence of noise that is confined to specific features, improving generalization, a valuable trait.

An associated concept in the realm of producing feature-reduced encodings is sparsity: only a small subset of neurons stays active when faced with various stimuli. This leads to improved energy efficiency [12], as well as more possible encoding permutations, allowing for more complex representations. Thus, comparable to the effects of disentanglement, sparse encoding may reduce noise interference by separating different features from each other in a meaningful way.

Computational models can be designed to behave similarly to the neuroscientific principles that inspire them. The next section provides an overview of computational models that are relevant to the Dev-AE approach.

## 2.2 Existing Computational Models of Neural Development

Autoencoders are a type of artificial neural network, which compress input data to a bottleneck, also known as latent space, before decompressing the information to attempt reconstructing the original input. The latent space houses a feature-reduced representation of the input, solely preserving the most dominant features. Training an autoencoder involves feeding the model input data and subsequently measuring how closely the reconstruction resembles the original. Throughout training, the model's internal parameters, its weights and biases, are adjusted iteratively based on how well it reconstructs the image. In this way, autoencoders learn the features most important for reconstruction [13].

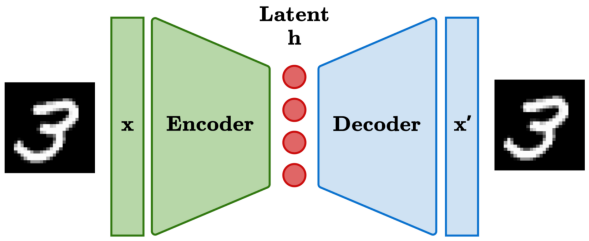



**Figure 2:** Architecture of a conventional autoencoder. The encoder compresses the input data $\mathbf{x}$ into the lower-dimensional latent representation $\mathbf{h}$. The decoder then produces a reconstruction $\mathbf{x}'$ from $\mathbf{h}$. The autoencoder minimizes the reconstruction error between $\mathbf{x}$ and $\mathbf{x}'$, ensuring the output closely approximates the original input.

While autoencoders have many applications, such as detecting anomalies, denoising data, and reducing file size, this research focuses on an autoencoder's dimensionality reduction, feature learning, and interpretability aspects [13]. For this analysis, autoencoders trained on images are used. Either simple or complex architectures can be employed, depending on the complexity of these images. In this thesis, the models belong to one of two architectures: multilayer perceptrons (MLPs) and convolutional neural networks (CNNs).

Both MLPs and CNNs use layers of neurons that pass information forward through the network: the input image moves through the encoder, the latent space, and lastly the decoder, after which the models return a reconstruction. The reconstruction is compared to its original counterpart, and the mean squared error (MSE), the average squared difference between the reconstructed and original pixels, is calculated. The reconstruction error is propagated backward through the network to update neuron weights [13]. In MLPs, each neuron is connected to all neurons in its adjacent layers. CNNs are more complex, using filters known as kernels to reduce certain pixel regions and extract important features. As this technique views pixels in groups, it retains the spatial aspect of the images.

The number of neurons in the latent space defines its dimensionality. In the conventional AE model, the size remains fixed. This thesis, however, revolves around understanding the Dev-AE model, where dimensionality increases over time [2]. Instead of a conventional, fixed latent space, the Dev-AE is initialized with a small bottleneck, meaning the latent space is restricted. This, in turn, enforces a lower-dimensional representation. As training goes on, the bottleneck is slowly expanded, which leads to a guided increase in dimensionality over time, mimicking the brain's development. While the Dev-AE and the brain share many concepts, often due to overlapping terminology, it is important to note that the Dev-AE models nothing more than the increase in dimensionality, as opposed to trying to provide a complex model of the brain. Following a brief literature review, this thesis will provide a detailed analysis of the effects that the increasing dimensionality of this model introduces.

The concept of manipulating the size of an artificial neural network has been explored before: *Self-Expanding Neural Networks* dynamically add layers to the model, once it begins

reaching congestion [14, 15]. This approach is successful in its way of finding an efficient network size, which is both energy-efficient and performant. Although the work regards regression networks, it is relevant as it shows that it is possible to train a model whose architecture changes during development. Another approach, *Adaptive Latent Dimensionality Variational Autoencoders*, begins with a large latent space and reduces this during training [16]. This was tested on variational autoencoders, which learn a probabilistic latent space by encoding inputs as distributions and are prone to overfitting when the latent space is too large. Here, the aim is to find an efficient bottleneck size dynamically by decreasing the size of the latent space in increments. This method was proven to be faster than conventional alternatives like grid search, which tests multiple models with different bottleneck sizes individually. Thus, this method was found to be more effective in finding an optimal bottleneck size.

Other autoencoder models involve introducing more complex approaches with the goal of specifically tuning certain aspects of the latent space. *$\beta$-VAEs* are variational autoencoders that introduce a $\beta$ variable, which can be used to tune the disentanglement explicitly [10]. This approach is successful at finding a separated latent space, but worsens reconstruction accuracy. This highlights an important trade-off: promoting disentanglement, which leads to better interpretability and generalization, may come at the cost of worse reconstructions. In response, the authors found a work-around. Relaxing the strength of the $\beta$ variable over time, promotes disentanglement first before improving reconstruction over the course of training [17]. Tuning sparsity has also been tried. *$k$-Sparse Autoencoders* retain only the $k$ bottleneck neurons with the largest activations, setting the rest to 0. This enforces a sparse latent space and was found to lead to better classification results [18].

In summary, there are existing approaches to investigate the capabilities of feature reduction in autoencoders. As explored above, some change the number of neurons or layers in the model to improve energy efficiency, while others explicitly tune disentanglement or sparsity. These traits are beneficial, leading to improved interpretability and generalization. On the whole, the literature review shows that small, sometimes simple modifications to a model can positively affect performance across varying tasks, especially when these changes occur dynamically during the model's training, adapting to the demands of the task at hand.

# 3 Methods

## 3.1 Architecture of Standard Autoencoder Models

To serve as a control group, standard autoencoder models with the same architecture as the Dev-AEs were used, differing only in the bottleneck layer, which remains fixed in AEs. The models implemented in this thesis are largely based on those from the original Dev-AE work [2] and [19], with modifications to aid interpretability and to adapt them to different datasets. The implementation of all models and methods can be found in the thesis' GitHub repository[1].

The MLP autoencoder was trained on the MNIST dataset, which consists of grayscale handwritten digits containing numbers from 0 to 9, corresponding to 10 classes [20] (see Figure 3a). Each image has 784 pixels, which the autoencoder reduces to 32 values in the bottleneck using five fully connected hidden layers. Each hidden layer applies the non-linear ReLU activation function. MNIST is a low-dimensional dataset, which means that, although each image has many pixel values, the information contained in the images can be represented with far fewer features [21].

In contrast, the CIFAR-10 dataset is comprised of higher-dimensional, natural images. These colorful, real-world photos show 10 different classes of objects, for example, an airplane, a bird, or a ship, introducing more complex patterns and a broader range of frequencies [22] (see Figure 3b). This dataset was used to train CNNs. The encoder and decoder, respectively, contain five

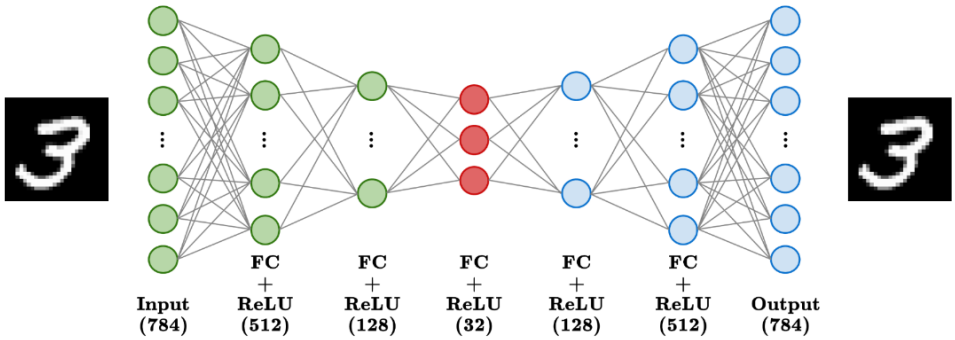
[1]Thesis GitHub Repository: https://github.com/DiJam20/understanding-developing-autoencoders.

hidden convolutional layers, all followed by ReLU activation functions, except for the final decoding layer, which uses a Tanh activation function. To keep a consistent bottleneck shape between the MLP and the CNN, the output of the final encoder becomes reshaped to a vector to provide a 1D latent space. The CNN's bottleneck reduces 3072 pixels to 128 values.

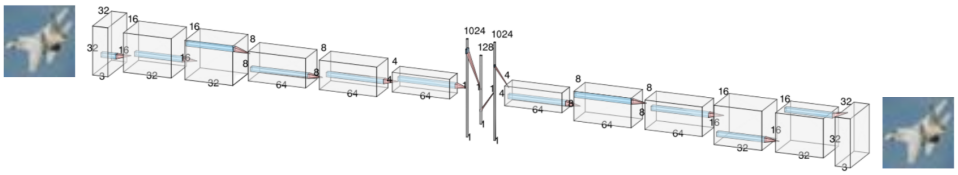
**(a)** MNIST **(b)** CIFAR-10

**Figure 3:** Example images from benchmark datasets used to train the autoencoders. (a) MNIST [20]: contains handwritten digits from 0 to 9 (28x28 grayscale images). (b) CIFAR-10 [22]: contains natural images from 10 different object classes (32x32 RGB images).

Both the MLP and CNN models are trained using stochastic gradient descent with a learning rate of 0.1, momentum of 0.9, and a batch size of 128. MSE was used as the loss function for gradient calculation and backpropagation. Model details are visualized in Figure 4 and further specified in the Appendix (1). In total, 40 AE and 40 Dev-AE models are trained on MNIST, and 10 of each model variant on CIFAR-10. All results and diagrams in this thesis present the average over those models. The work in this thesis contributed to a co-authored paper, currently under review [23], which found the same results with 40 CIFAR-10 models, confirming that findings remain consistent when scaled to a larger number of models.




**(a)**



**(b)**

**Figure 4:** Neural network architectures for autoencoder models. (a) Multilayer perceptron autoencoder trained on MNIST. (b) Convolutional autoencoder trained on CIFAR-10 [24]

### 3.2 Overview of the Developing Autoencoder Model

Having already introduced the AEs, few additions are needed to describe the architecture of the Dev-AEs. Again, the implementation is largely adopted from the previous work [2], [7]. Both MLPs and CNNs take their architecture, hyperparameters, and optimizers, from the AE models, differing only in the bottleneck size. Rather than the bottleneck being fixed, it is incrementally increased during training. Beginning with a low number of neurons, it grows to the same size that the AE started with (see Figure 5).

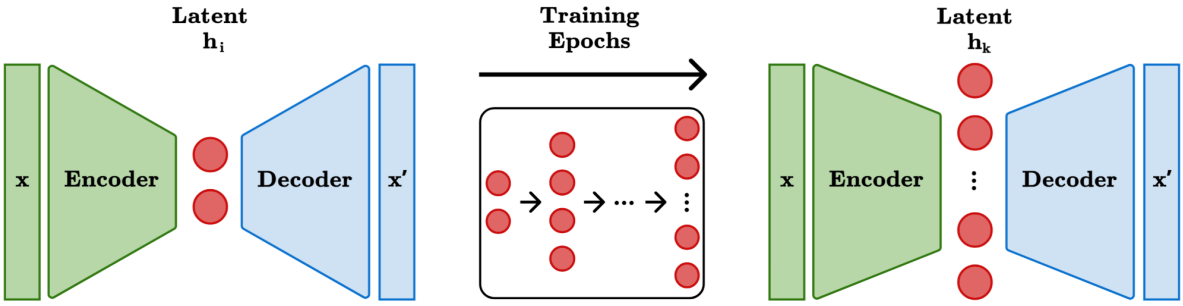



**Figure 5:** Training process of a Developing Autoencoder. The model is initialized with a small latent space $\mathbf{h_i}$, which progressively grows to $\mathbf{h_k}$ throughout its training.

Like in the AE, the weights and biases of the first bottleneck neurons in the Dev-AE are drawn from a standard normal distribution. The first increase arrives after training the initial architecture for a few epochs. The existing neurons in the bottleneck are also further optimized, and not frozen during training. For new neurons, initialization occurs through cloning multiples of previous neurons and adding noise to these cloned weights. The noise follows a Gaussian distribution with mean and 1/10th standard deviation of the existing neurons' statistics. The remaining neurons are entirely initialized with noise sampled from a Gaussian distribution with mean and standard deviation matched to the existing neurons' statistics. While the models used in this analysis almost exclusively add noise in small increments, which employ only the latter initialization method, the cloning method deals with cases of large increases. Indeed, if very large numbers of neurons are introduced at the same time, the effect on the existing network is destructive, unfavorably impacting performance. Cloning combats this by giving neurons a better starting point, allowing for a more consistent performance during development.

To display the methodology, the formula for introducing new neurons into the bottleneck is presented. The old bottleneck with $N$ neurons is denoted as $\mathbf{h_{old}} \in \mathbb{R}^N$. This is expanded to a size of $M$ neurons denoted as $\mathbf{h_{new}} \in \mathbb{R}^M$. The initialization of the new neurons is as follows:

$$\mathbf{h_{new},}_i = \begin{cases} \mathbf{h_{old}}, i & \text{for } 1 \leq i \leq N, \\ \mathbf{h_{old}}_{,(i-1) \bmod N+1} + \epsilon_i & \text{for } N < i \leq N + k \cdot N, \\ \eta_i & \text{for } N + k \cdot N < i \leq M. \end{cases} \quad [7]$$

where:

- $k = \left\lfloor \frac{M-N}{N} \right\rfloor$ is the number of times each original neuron is cloned,
- $\epsilon_i \sim \mathcal{N}(\mu_{\text{old}}, \frac{\sigma^2_{\text{old}}}{10})$ cloning noise, matches existing neuron statistics with weakened variance,
- $\eta_i \sim \mathcal{N}(\mu_{\text{old}}, \sigma^2_{\text{old}})$ initialization noise, matches existing neuron statistics,
- $(i-1) \bmod N + 1$, maps indices $i > N$ to the original neurons 1, 2, ..., $N$.

This thesis further expands the developing model with a convergence criterion. In order to determine when to increase the bottleneck size, previous work passed an additional list of values to define how long each bottleneck size is to be trained on. The convergence method provides a more dynamic growth, similar to approaches introduced in Section 2.2 ([14], [15], [16], and [18]). Using the convergence approach, the model must be presented with only one set of bottleneck sizes. If the training loss is only 0.1% different to the loss of the previous epoch, the model moves onto the next bottleneck size. The list of bottleneck sizes that were iterated through for the MLP model was [4, 10, 16, 24, 32], and for the convolutional model was [6, 10, 16, 28, 48, 90, 128]. Since analysis required autoencoders to be of a similar architecture, this approach was used solely in the exploratory phase to find an efficient set of parameters, while the analysis itself used the old approach with the dynamically discovered parameters.

An important concept that emerges from the Dev-AE setup is the distinction between groups of neurons introduced at different stages during training. Neurons introduced at the same time during training of the Dev-AE are termed "neuron groups". While certain groups exist from the start, others are initialized much later. Some of these neuron groups contain a larger number of neurons than others, which is accounted for in the following comparisons.

### 3.3 Validating Dimensionality Growth in the Models

The dimensionality of the latent space is broadly linked to the size of the latent space. Specifically, the number of neurons provides an upper bound for the dimensionality. To measure dimensionality during development more precisely, a dimensionality estimation is used. One should note that this measure does not rise boundlessly with increasing amounts of neurons, and is instead limited by the complexity of the dataset. Making use of a dimensionality estimation method allows for the verification of whether an incremental increase in the bottleneck is meaningful, because the bottleneck must be functionally constricted, in order to limit dimensionality. The Two Nearest Neighbors (TwoNN) method measures the distance to the first and second nearest neighbors of each data point. Next, the ratio of these two distances is calculated and averaged across all data points, which provides an estimate for how far apart the points are, and thus how many dimensions matter [25]. For this reason, this thesis will use TwoNN on the bottleneck activations of test images in order to examine the model's dimensionality growth.

### 3.4 Principal Component Analysis

Principal Component Analysis (PCA) is a common linear dimensionality reduction method, which finds the principal components. These are the orthogonal directions that capture the most variance. These components are ordered from capturing the highest to lowest amount of variance. When setting the number of components to $n$ the $n$-most variance explaining components are calculated [26]. This measure will be used to determine which features neurons respond to, as well as providing an analysis of the latent space's stability throughout development.

### 3.5 Receptive Fields

In Section 2.1, biological receptive fields were introduced as the external stimulus that excites a neuron. While receptive fields in artificial neural networks, especially convolutional ones, refer to the input region that a neuron is influenced by, this work adopts the neuroscientific definition. Here, artificial receptive fields are defined as the type of input that activates a neuron most. In the autoencoder model this concept can be extended, by not only finding a data point that excites a neuron most, but by generating a new image, which maximally activates a neuron. This reveals which features a neuron responds to most. The technique used to achieve this is called activation maximization [27]:

$$\mathbf{f} \leftarrow r_\theta \left( \mathbf{f} + \lambda \frac{\partial a_i}{\partial \mathbf{f}} \right) \quad \text{where: } r_\theta(\mathbf{f}) = (1 - \theta_{\text{decay}}) \cdot \mathbf{f} \qquad [27]$$

where:

- $\mathbf{f}$ is the receptive field being iterated,
- $a_i$ is the activation of the i-th neuron in the bottleneck,
- $\lambda \frac{\partial a_i}{\partial \mathbf{f}}$ represents the gradient ascent step, which adjusts $\mathbf{f}$ so that $a_i$ is increased, scaled by the learning rate $\lambda$,
- $r_\theta$ is the regularization term with $\theta_{\text{decay}}$ set to 0.1. L2 decay is used, which penalizes large values and tends to prevent a small number of extreme pixel values from dominating the example image.

Though receptive fields can be analyzed visually, the frequencies present in the receptive fields can be examined to gain a better understanding of underlying characteristics, in turn revealing what frequency a particular neuron is most responsive to. For this purpose, the Fast Fourier Transform performs a Fourier decomposition, transforming a receptive field into the frequency domain. When working with CIFAR-10, receptive fields are converted to grayscale in order to find average frequencies across all channels. To obtain the value of the power associated with each frequency, the squared magnitude of the Fourier transform is calculated. To visualize the average power of each frequency, a radial profile is used, binning the frequency components by their radial distance from the center. Averaging the receptive field power spectra across neurons within each group reveals the frequencies captured by different neuron groups.

### 3.6 Evaluating the Stability of Learned Features

Determining the stability of features learned in earlier neuron groups throughout development is key in showing the effectiveness of the Dev-AE. This helps to assess whether neuron groups in the Dev-AE learn and maintain specific features during the entirety of development, or whether those specific representations are temporary, thereby making earlier neuron groups indistinguishable from AE neurons over time. To evaluate this, one can use measure stability of two concepts: principal components of the latent space and receptive fields of the bottleneck neurons. Both are calculated after every epoch. The difference between each epoch and the final epoch is then calculated using the cosine angle difference.

#### 3.6.1 Stability of Principal Components

After each training epoch, test images are fed into the models to get corresponding encodings, on which a PCA is performed. The number of principal components calculated is equal to the number of neurons in the bottleneck $N_{\text{current}}$, as PCA cannot find more components than the current number of dimensions. For the Dev-AE, the earlier training epochs have smaller bottlenecks, which result in fewer principal components. To compare the principal components of each epoch to the final epochs, the missing values are zero-padded. Comparing principal components provides a clear view of how the main features captured by the encodings develop over time.

### 3.6.2 Stability of Receptive Fields

As outlined in Section 3.5, receptive fields are computed individually for each neuron. This means that, for every training epoch, a receptive field is computed for every neuron in the bottleneck. To compare these images using the cosine angle difference, each receptive field is flattened to a one-dimensional vector.

After computation, the angle differences between each epoch and the final epoch are displayed as a heatmap. The values padded with zeros are ignored, resulting in a terracing form that is characteristic to the Dev-AE, since the AE does not require any padding. In the heatmap, the slow but sure shift of the principal components and the receptive fields towards their final state is represented by a smooth color gradient. If this trend is interrupted by spikes of high angle difference, especially as later neuron groups get introduced in the Dev-AE, it signals that early neurons are not sticking to their learned features.

## 3.7 Noise-Based Perturbation Methods

### 3.7.1 Introducing Principal Component Noise

To evaluate how individual neurons respond to specific PCs in the input images, a perturbation method is applied, where Gaussian noise is introduced into select groups of principal components. First, each image is reduced using PCA, followed by the addition of Gaussian noise to principal components with the same indices as the indices of neurons in a neuron group. Matching principal component indices to the neurons that were introduced together allows for an analysis of the group as a whole. The noisy principal component representations are reconstructed using an inverse PCA, before being used as inputs for the autoencoders. Then the differences in neural activations in the bottleneck layer are measured. For each neuron group, one noisy reconstructed image is generated per original image. All of these newly produced images are passed into the models and activation across all neurons is measured. To visualize which principal component noise has the greatest impact on which neuron group, the data is plotted by rank of highest perturbation. This process enables the identification of principal components, or features, that specific neurons respond to, from highest to lowest.

### 3.7.2 Introducing Latent Space Noise

To identify which neuron groups are the most important for a good decoding, the encodings of specific groups are perturbed and the resulting reconstruction loss is analyzed. First, images are fed into the models and the subsequent encodings are collected. Gaussian noise is introduced to each neuron group. This noise is matched to the mean and standard deviation of the neuron group's original activations, but the standard deviation is amplified by the reciprocal of the number of neurons in that group. This ensures that the amount of disturbance to a group of neurons is proportional to the group's inherent variation and size, which guarantees that the effect of the added noise is not simply stronger, because of it being added to a group with more neurons or lower overall activation. The perturbed latent representation is passed through the decoder to generate a reconstructed image, which is used when calculating loss. Adding noise in this targeted manner helps quantify which neuron groups encode the most important information for image reconstruction, and allow the visualization of which features get destroyed in the reconstructions.

### 3.7.3 Introducing Different Frequency Noise Types

Measuring the impact of different frequency noise types allows the identification of which types of noise the models are robust against. To add noise in specific frequency ranges, the first step

is to generate noise in the dimension of an input image. This noise is transformed into the frequency domain via a Fourier transform and shifted, so that the zero frequency component is in the center. This way, the lowest frequencies are represented by the center of the image, with rings with a larger radius representing the higher frequencies. By creating a ring mask with a specific inner and outer radius, only certain radial frequencies of the noise are kept. By adding frequency noise to each image in the dataset, a new noisy dataset is created for low-, medium-, and high-frequencies. Rings of radius 0–3 produce low, 4–7 generate medium, and 8–16 create high frequency noise. This noise is brought to the spatial domain using an inverse Fourier transform and can simply be added in the spatial domain. This is due to the linearity of the Fourier transform: spatial domain addition is mathematically equivalent to adding noise directly in the frequency domain.

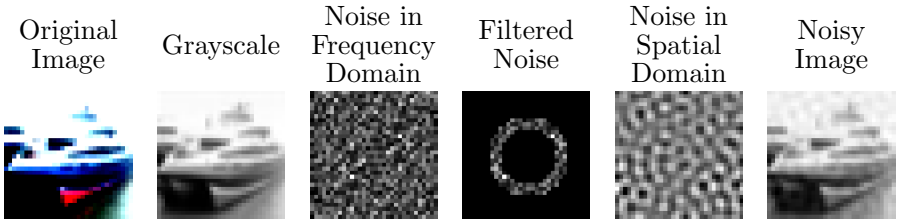



**Figure 6:** Generation of images with noise added in the frequency domain. The approach is shown for CIFAR-10, but after the conversion to grayscale, the process for MNIST is analogous. Noise is generated and converted to the frequency domain using the Fourier transform and shifted, so the zero frequency is at the center. A filter is applied, leaving only select frequencies. This noise is inverse transformed to the spatial domain where it is added to the original image.

Depending on the frequency filter, a different number of pixels is being manipulated, as the higher circumference will yield a larger area. To account for this, the amount of noise is normalized via dividing by the magnitude of the noise, resulting in a unit vector. This can then be scaled with any desired value. In order to find out how well the encodings of the noisy images can be classified, a logistic regression classifier is trained on the models' encodings of clean input data. The classifier yields a classification accuracy for clean images for control, and accuracies for low-, medium-, and high-frequency noise. Then a t-test is used to check for significance between the accuracy of the AE and Dev-AE. A value of $p < 0.05$ is marked with `*`, and non-significant results are labeled with `ns`.

### 3.8 Determining Neuron Influence for Classification

To determine which neuron groups are most influential for classification, the coefficients of a trained classifier can be analyzed. A logistic regression classifier is trained on the encodings that are collected from feeding images through the models. Since both MNIST and CIFAR-10 contain 10 classes, a multiclass classifier is used. Coefficients in a multiclass classifier are positive when predicting the positive class and negative when predicting the negative class. The absolute value is taken, and the average over all classes signifies how much each neuron influences the prediction. Because each value in the encodings corresponds to one neuron in the bottleneck, examining the coefficients of the classifier reveals, which bottleneck values have the greatest impact on classification. The higher its coefficient, the more a neuron influences the classification. By grouping neurons according to their neuron group, it is possible to identify which neuron group is most important.

### 3.9 Measuring Sparsity in Neural Activations

To measure sparsity, the average neural activation and the number of inactive neurons is measured. First, these measures are analyzed in the latent space, as this directly answers the research question. Secondly, the two measures are applied to all hidden layers, to see if the effects in the bottleneck layer are transferred to neighboring layers.

When calculating average activation per image, zeros are excluded to prioritize the average activation of active neurons. The average activation reveals if there are neurons with consistent high activations, which thereby indicate that these neurons are encoding important features that are present in most images. At the same time, the average activation highlights neurons that only have low activations, without being zero. Activation can also be averaged per image, showing if an image contains features learned by the neurons and thus activating them.

The other and more classic approach for measuring sparsity is to find the percentage of inactive neurons in the encodings. For the MLP architecture, a ReLU activation function is applied right after the bottleneck, and is thus applied before the encoding is extracted, prompting many of the neurons to exhibit zero activation. Meanwhile, the CNN uses a ReLU function after the bottleneck has been reshaped (see Table 2), instead of directly after the bottleneck layer. This choice was made as it yielded better results in downstream tasks, like receptive field visualization and interpretability. Because zero activations cannot be counted directly, all activations under a threshold of 1e-4 are counted.

Based on all the activations a neuron produces, given 10,000 test images, sparsity is measured in all hidden layer neurons of the model and each neuron is classed as inactive, conditionally active, or universally active. The previous measurement of inactive neurons can be simplified to always counting pure zeros, as the hidden layers all use ReLU. Neurons with activations that are zero with all test images are termed inactive. Those that get activated to some degree by all presented test images are sorted as universally active, which indicates that they have learned features represented in every image. The remaining neurons that activate occasionally are called conditionally active.

## 4 Results

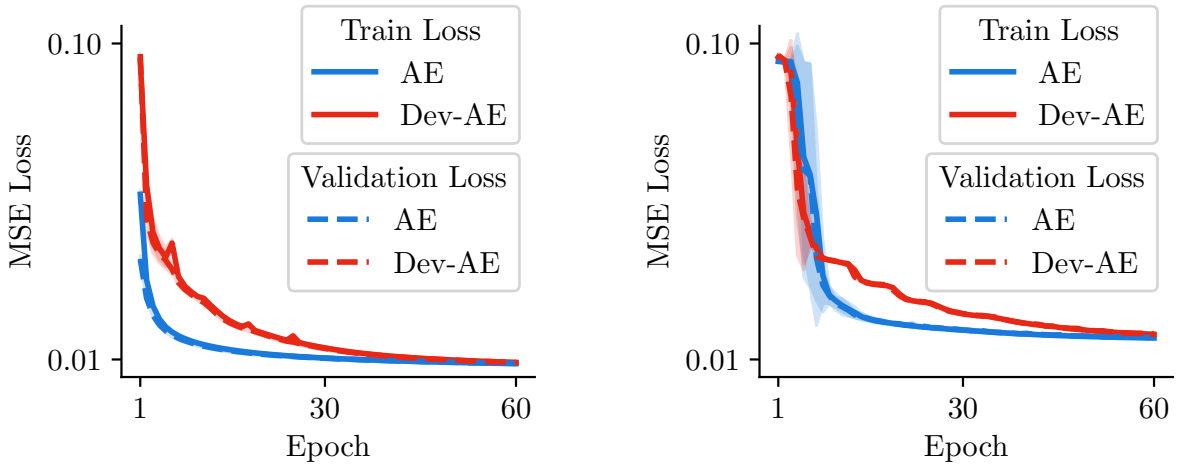
Starting out, a comparison of performance metrics across all models serves to show that the models come from a similar base background, hence permitting fair comparison. Following this, key findings are grouped in the sections, disentanglement, ordering of importance, and sparsity.

### 4.1 Model Performance Metrics

As the main focus of the past Dev-AE paper [2] was performance comparison, only a brief performance summary will be presented, covering the results of reproducing this work with new data, tweaked hyperparameters, and the novel converging method.

#### 4.1.1 MSE Loss and Reconstruction Performance

Training and validation loss curves are the most common performance measures for autoencoders, showing how well the reconstructed image mirrors the original image. For both datasets, the AE converges earlier while the Dev-AE takes longer to reach the same loss. Importantly, the Dev-AE does eventually reach the same loss, laying a solid foundation for a latent space comparison. It is unsurprising that the Dev-AE, with its smaller early latent space size, requires longer training. Visibly evident is the convergence approach, where the Dev-AE's bottleneck size increases upon training loss convergence. As the models are learning to handle the newly introduced neuron group, one can observe a slight decrease in performance in the epochs when

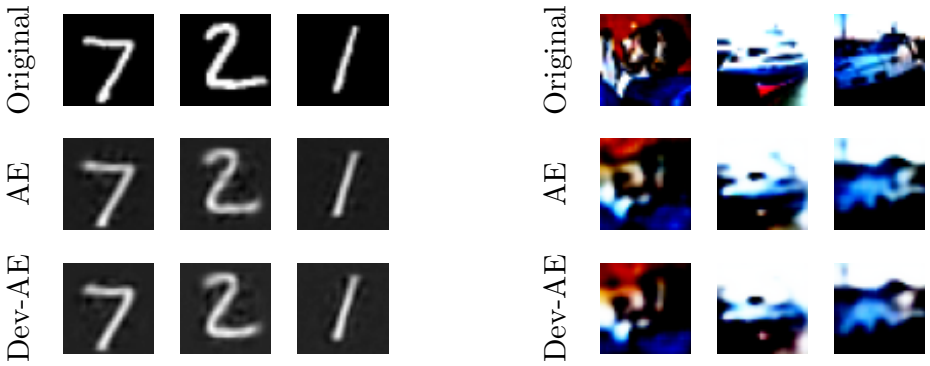



**(a)** MNIST **(b)** CIFAR-10

**Figure 7:** Comparison of the training and validation loss curves for AE and Dev-AE models trained on the MNIST (a) and CIFAR-10 (b) datasets.

the new noisy neurons are introduced. However, the MSE loss quickly returns to the downwards trend, demonstrating that the training on the earlier neurons is not lost.

The examples of reconstructions taken from the fully trained models also visualize that the reconstruction performance for both models is comparable.



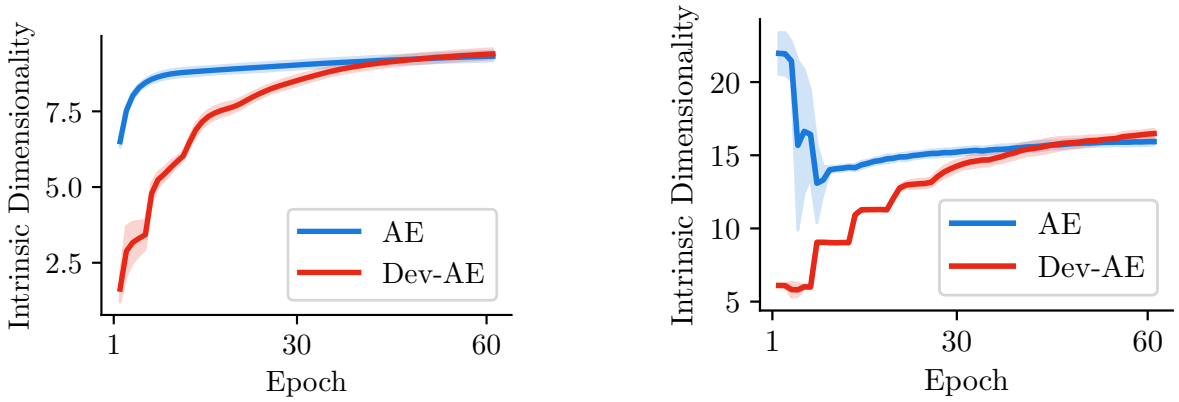

**(a)** MNIST **(b)** CIFAR-10

**Figure 8:** Examples of reconstructed images of AE and Dev-AE models trained on MNIST (a) and on CIFAR-10 (b) datasets.

### 4.1.2 Latent Space Dimensionality

Returning to the original description of the brain, the increase in dimensionality was described. This was the core feature that was meant to be modeled by the Dev-AE. To test this, the TwoNN method is used on the latent space representations, as described in Section 3.3.

Figure 9 validates the guided dimensionality increase that is trying to be modeled. The dimensionality increases slightly in the AE too, but not in the same slow manner of the Dev-AE. The final dimensionality of both models is comparable, however, the CIFAR-10 Dev-AE finds a slightly higher dimensionality, which can lead to better classification performance downstream. In the CIFAR-10 AE the dimensionality begins high, an artifact of the noisy initialization of all 128 initial bottleneck neurons. This randomness spreads across dimensions, but the effect goes




**(a)** MNIST **(b)** CIFAR-10

**Figure 9:** Development of intrinsic dimensionality in encodings during autoencoder training shows a guided increase of latent dimensionality in the Dev-AE. The Two Nearest Neighbor method [25] estimates the dimensionality of the models trained on MNIST (a) and on CIFAR-10 (b) datasets.

away as the model learns more meaningful representations. It is also notable that the overall intrinsic dimensionality of both datasets is low, begging the question whether similar results would have been found with an even narrower latent space.

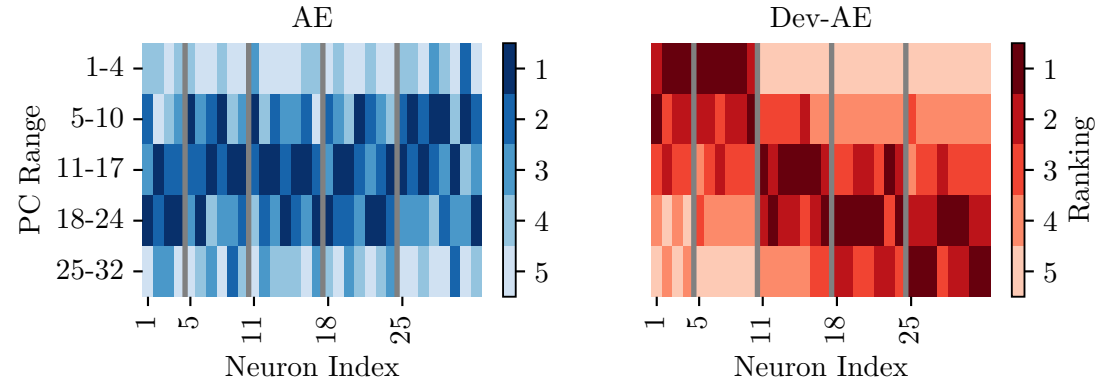
## 4.2 Disentanglement of Learned Representations

Disentanglement in the latent space describes the property of an autoencoder's bottleneck neurons to learn separated representations. Instead of every neuron learning a mixture of all features, each neuron responds to a certain feature of the input images. Though less mathematically defined, similar traits are found in the brain where neurons specialize on distinct features during development. The question then is: will the Dev-AE model show such patterns too. In contrast to the hands-off approach of the Dev-AE, $\beta$-VAEs [10] control disentanglement explicitly. These models sacrifice some reconstruction quality, but manage to successfully learn disentangled representations. Two methods are carried out to determine the learned features of the neurons in the Dev-AE models: finding the principal components that neurons respond to, and looking at what frequencies are most represented in their receptive field.

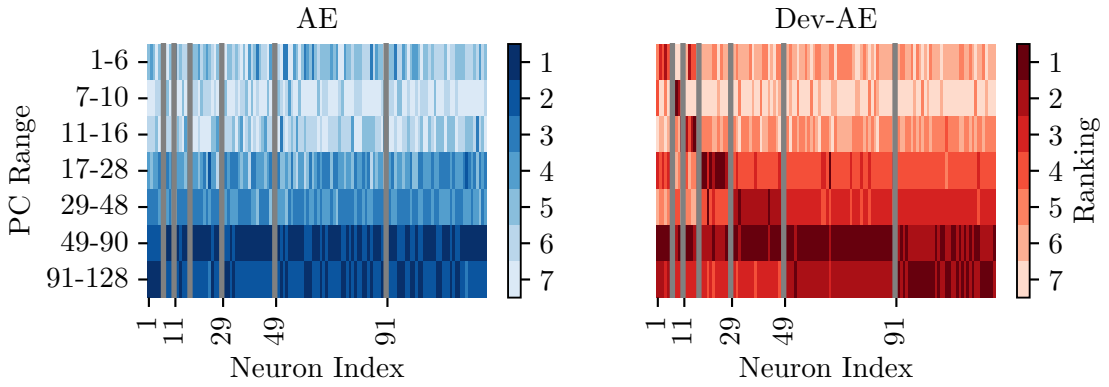
### 4.2.1 Disentanglement based on Image Principal Components

As described in detail in Section 3.4, the principal components are the orthogonal directions with highest variance in the data. Due to their orthogonality, the principal components represent uncorrelated, separated features. Accordingly, measuring if the neurons are learning different principal components provides a good indication of whether the neurons themselves are learning separated features. To measure this, noise is introduced in the principal components (see Section 3.7.1) of the input data, and the change in neural activation is measured.

A pattern arises in Figure 10 in both MNIST and CIFAR-10 models. All AE bottleneck neurons react similarly to manipulations of different principal components. The higher impact of some groups of principal components can be attributed to the larger group of principal components being manipulated. Altogether, this indicates that the AE models learn an entangled representation of the features that the principal components represent. To this, the Dev-AE forms a stark contrast. Noise in the leading principal components affects early neurons, while later neurons respond to lower principal components. Though a few deviations occur, causing




**(a)** MNIST




**(b)** CIFAR-10

**Figure 10:** Impact of noise added to sets of principal components (PCs) on the neural activation of each bottleneck neuron, ranked from highest to lowest impact. PC ranges match the sizes of neuron groups. Early neuron groups in Dev-AE models capture leading PCs, later groups capture lower PCs, in models trained on MNIST (a) and CIFAR-10 (b) datasets. This trend is not mirrored in the AE models.

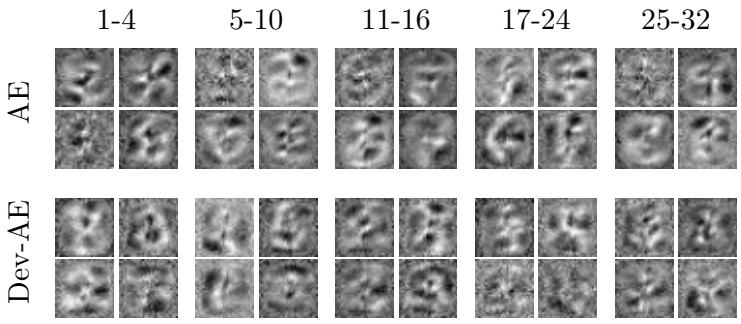
some neurons to be most impacted by an unexpected PC group, this can simply be traced back to the differences in group size and the randomness of the added noise.

The entangled manner in which AE bottleneck neurons learn principal components suggests that the same temporarily goes for the Dev-AE, as the latent space is fixed for short, intermittent periods of time. The earliest neurons learn a set of principal components, with representation within this specific group being entangled. However, disentanglement does exist across neuron groups, with each group learning a different set of principal components. This causes functional differences between the models. If noise of a specific feature (represented by one principal component) perturbs the image, only a subset of Dev-AE neurons is affected, whereas all neurons in the AE would be affected in the same scenario.

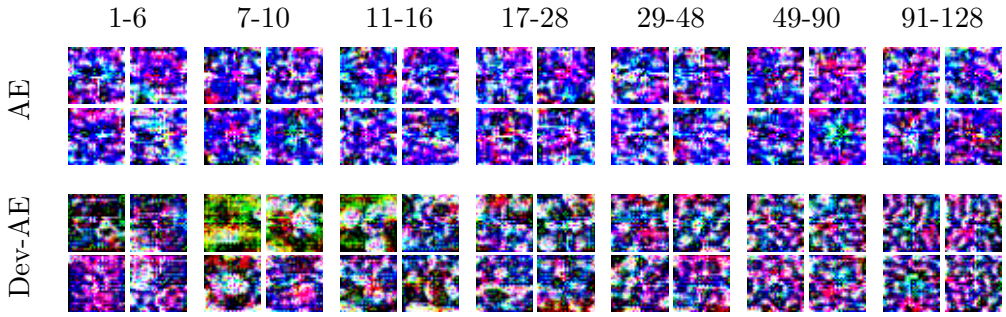
### 4.2.2 Disentanglement based on Receptive Fields

Another way to show whether neurons are learning different features is by looking at their receptive fields. These are the images that each neuron is maximally activated by, found using the activation maximization technique [27], as described in Section 3.5. Plotting examples of receptive fields of one model from each neuron group provides an overview of whether neurons are learning similar or different features.

A vital distinction must be made between the models trained on MNIST and those trained on CIFAR-10: while the natural images of the CIFAR-10 dataset have a wide range of frequencies,




**(a)** MNIST




**(b)** CIFAR-10

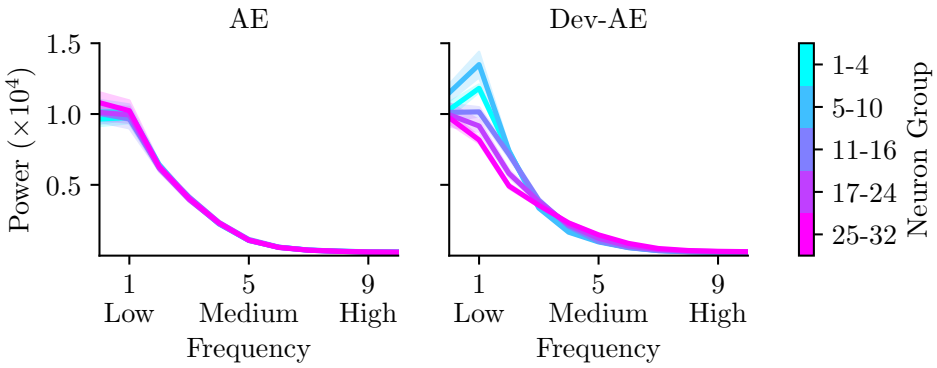
**Figure 11:** Examples of receptive fields of bottleneck neurons arranged by neuron groups. Receptive fields are calculated using the activation maximization method [27] and visualize which features maximally activate the neurons in the bottleneck. The Dev-AE trained on CIFAR-10 (b) exhibits a coarse-to-fine trend from early to late, while the AE exhibits a mixed learning of features. The models trained on the low-dimensional MNIST dataset (a) show little variation.

the handwritten digits of MNIST have a very small frequency spectrum. Therefore the results of MNIST are less relevant and neither aid nor hinder the claims made from the analysis of the CIFAR-10 plots.

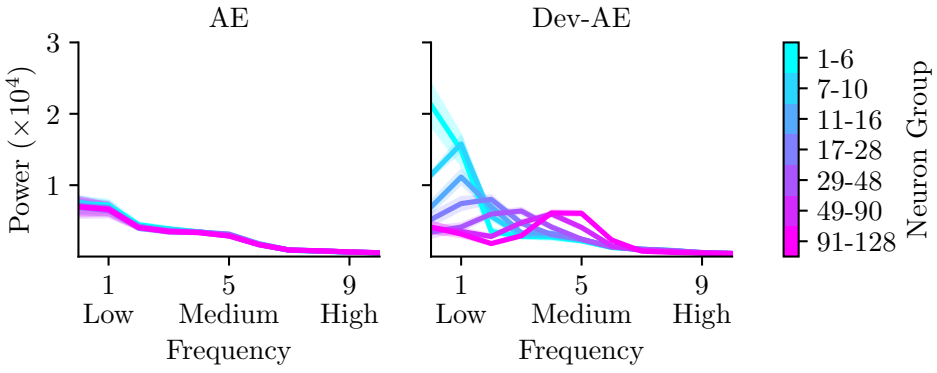
Though it is admittedly difficult to decrypt which exact features are being learned in the CIFAR-10 plots, there is a definite difference between the receptive fields of the AE and those of the Dev-AE. Specifically, the receptive fields of the AE all look similar, while the receptive fields of the Dev-AE seem structurally similar within a neuron group, differing rather across groups. Earlier neuron groups show larger, broad shapes that correspond more to low frequencies, while later neuron groups appear noisy, showing small, detailed shapes that are associated with high frequencies. To verify this claim, this thesis employs power spectra (see Section 3.5). From these, the frequencies that are most represented in the images can be deduced.

To reiterate, as neurons are unable to learn different frequencies without having a range of frequencies, MNIST results show only minute differences. However, the CIFAR-10 plots show a large difference between learned frequencies. Early neurons show large low-frequency values, with simultaneously less power in all other frequencies. Neuron groups coming directly after show less power in the very early frequencies, but a wider range of frequency, ranging from low to medium. In the final neuron group, frequency power peaks at around at a medium frequency, and ranges further into the higher frequencies. Non-grouped power spectra can be found in the Appendix (see Figure 25).

These observations fit the visuals of the example receptive fields, with broader shapes representing low frequencies in the early neurons and detailed shapes representing high frequencies in




**(a)** MNIST




**(b)** CIFAR-10

**Figure 12:** Power spectra of the receptive fields averaged by neuron group. Regular autoencoder (AE) models show no frequency separation between neuron groups. In contrast, Dev-AE models show separation, especially when trained on CIFAR-10 (b), and to a lesser extent on MNIST (a); early neuron groups capture low frequencies, and later groups capture progressively higher frequencies.

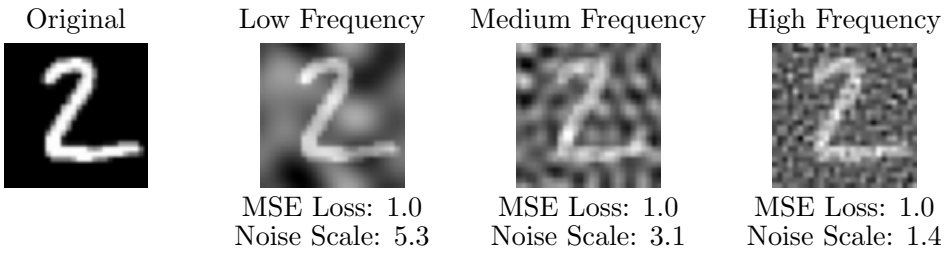
the later neurons. This also coincides with the analysis of learned principal components. When a PCA performs a feature reduction, the principal components are ordered by variance, and the top principal components are often general, broad shapes, while the later principal components tend to capture more detailed features.

### 4.2.3 Classification of Different Frequency Noise Types

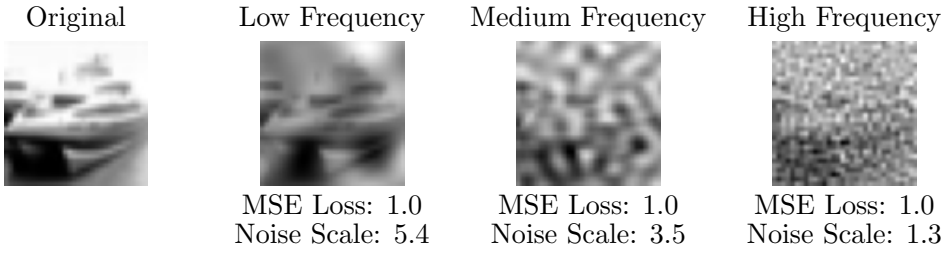
Adding noise to principal components proved effective to show disentanglement, but the noise added is more artificial than other types of noise that might be more likely to be encountered in real world data. Now that disentanglement of frequencies across different neuron groups has been shown, further testing involves noise being added to the low, medium, and high frequencies of the images. Being robust against these types of noise is a better measure of generalization, a term referring to the trait of a neural network performing well on unseen data. This often goes hand in hand with disentanglement: if some features change most neurons remain unaffected.

The process involved generating noise, filtering out the noise with the relevant frequency, before adding this filtered version back to the images (see Section 3.7.3). To accurately measure which type of frequency noise most affects the classification, the amount of added noise must be consistent. This way, any differences in accuracy are truly linked to the stronger effect of certain frequencies, as opposed to merely being an effect of the size of the filter used for each frequency

range. Examples in Figure 13 show the varying noise scales added to different frequencies, resulting in the same MSE loss compared to the original image.




**(a)** MNIST



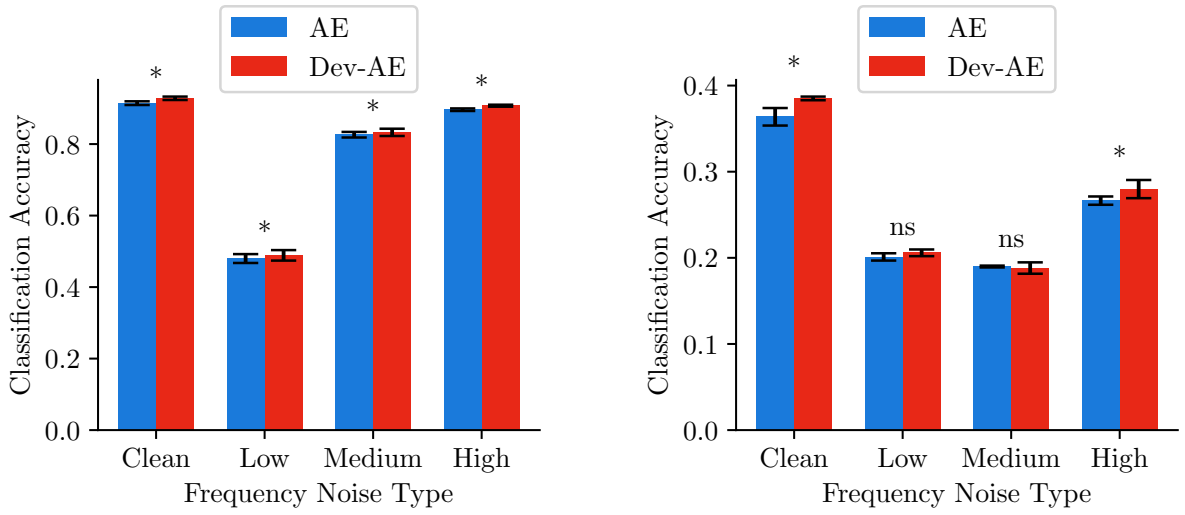

**(b)** CIFAR-10

**Figure 13:** Examples of MNIST (a) and CIFAR-10 (b) images with low-, medium-, and high-frequency noise added, where the strength of the noise is normalized to ensure equal MSE loss. Loss is calculated between each noisy image and its original counterpart. The noise scale indicates how much noise of a specific frequency must be added in the frequency domain to achieve an MSE of 1.0.

The example images show striking visual differences. Low frequency perturbations result in broad features being manipulated, leading to smudging in the image. Meanwhile, the high frequency noise perturbs finer details like edges. Having validated that the normalization yields identical MSE loss and having shown what images are being worked with, this work now shows how well a classifier trained on clean encodings responds to encodings from noisy inputs.

Notably, the clean images show differences in classification accuracy. The classifier is significantly better at classifying the clean encodings of the Dev-AE for both datasets. This is possibly due to the disentanglement, through which the Dev-AE's encodings become better separated and interpretable to the classifier. Overall classification accuracy in the AE and Dev-AE is good in MNIST at high and medium levels, while only at a high-frequency level for CIFAR-10. This aligns with the findings of the power spectra. Most features of MNIST align with low frequencies, while in CIFAR-10 the features have a wider frequency range, with high frequency being represented least. The classification therefore is most robust against the frequencies that are least present in the dataset. The better classification of the clean images transfers to most other frequency types too. This indicates that the CIFAR-10 Dev-AE has a stronger robustness against high frequency noise, and against all noise types for the MNIST Dev-AE.

So long as the frequency of the dataset permits it, both the principal component and the receptive field measures show evidence of disentanglement solely in the Dev-AE. Disentanglement in the latent space of an autoencoder is a desirable trait, as it enhances interpretability, since certain sets of neurons control certain sets of features. Interpretability is especially helpful for answering the research question to gain a deeper understanding of the models. Furthermore, the measuring of classification accuracy shows that the Dev-AE's disentangled encodings perform better on classification tasks, and are more robust against at least high-frequency noise.




**(a)** MNIST **(b)** CIFAR-10

**Figure 14:** Classification accuracy of a logistic regression classifier trained on clean encodings and evaluated on encodings from low-, medium-, or high-frequency noisy images. Differences between the AE and the Dev-AE are labeled with the results of a t-test, where a $*$ represents a statistically significant result ($p < 0.05$). The Dev-AE models achieve significantly higher classification accuracy. Additionally, best classification accuracies are achieved on noise of frequencies least present in the original data: medium and high frequencies for MNIST (a) and high frequencies for CIFAR-10 (b).

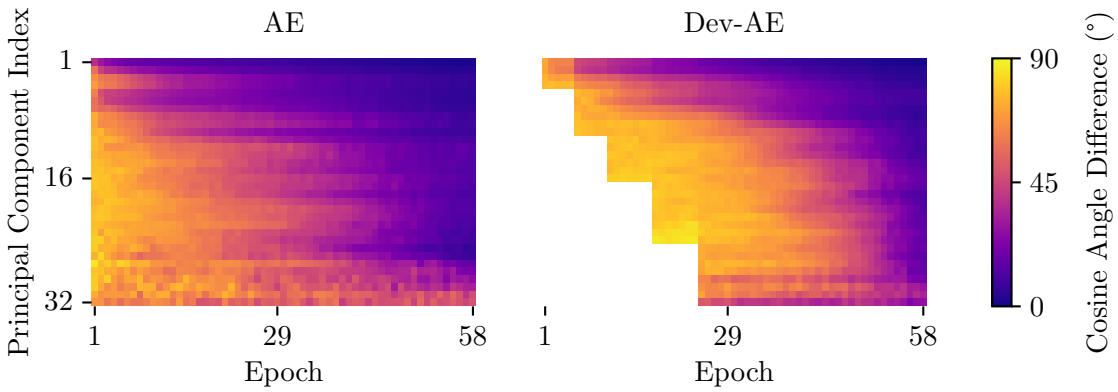
### 4.3 Hierarchical Ordering of Importance

So far, disentanglement has been shown to exist in the Dev-AE across neuron groups. However, the next question one may ask is whether the neurons in a Dev-AE show an ordering of importance. For example, are the most important features being learned by the first neuron groups, followed by the progressively less important features. In a way, this has already been answered by the analysis of which neurons learn to respond to which principal components in Section 4.2.1. This showed how earlier neurons were indeed learning leading principal components. In addition to this feature importance measure, the following two measures can be verified: importance for reconstruction and importance for classification. Before beginning this endeavor, it is necessary to measure whether the neurons are sticking to the information that they learn. Once this is confirmed, the two importance measures can be measured on the fully trained models. The results from that then transfer to earlier steps in development. To show if neurons maintain their learnings over time, stability is measured.

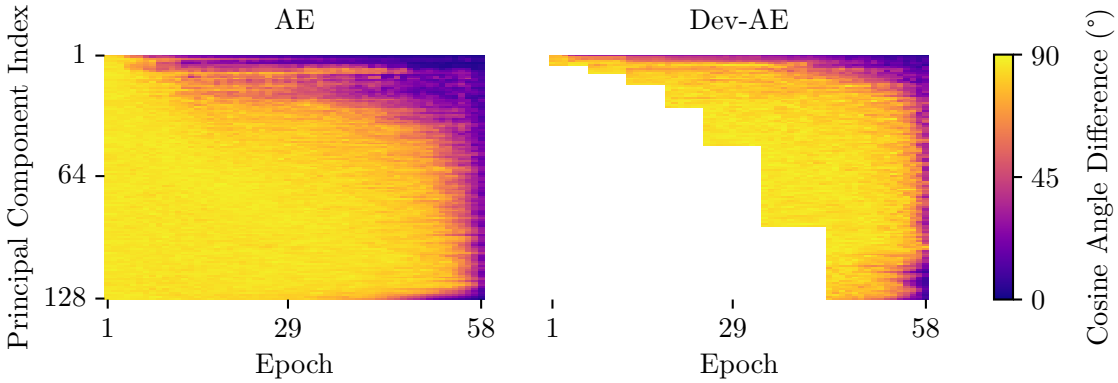
#### 4.3.1 Stability of Principal Components and Receptive Fields

The stability of learned features in the bottleneck neurons is measured using the principal components of encodings, as well as every neuron's receptive field, during each step of training. This shows how the latent space is organized and how long it takes, if at all, for features representations to stabilize. This is measured using the cosine angle difference between each epoch and the final epoch. The findings also hold when measuring the difference between consecutive epochs, as shown in the Appendix (see Figures 26, 27).

Comparing the principal components over time demonstrates multiple trends in the structure of the latent space during its development. To begin, one can see that the leading principal components are the ones to stabilize first in the AE, and remain quite stable throughout development. Later principal components take longer to stabilize, sometimes showing heightened




**(a)** MNIST



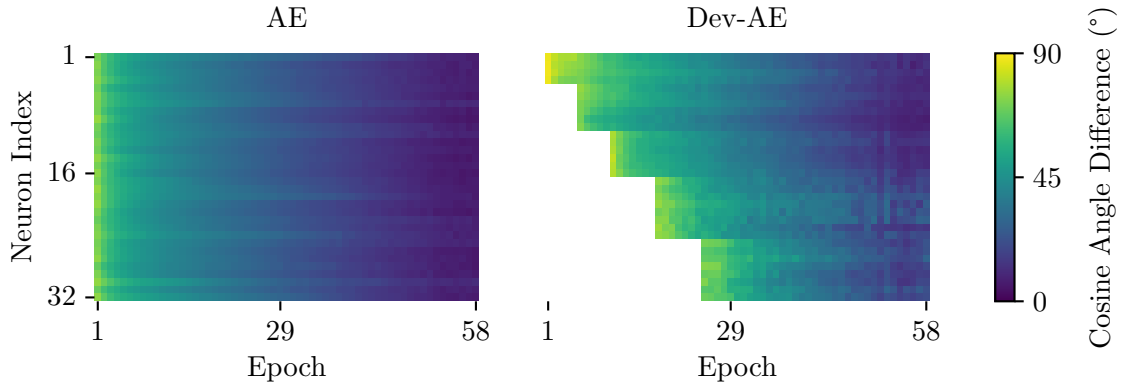

**(b)** CIFAR-10

**Figure 15:** Stability of the principal components (PCs) in the encodings. Cosine angle difference is measured between a given PC at a specific epoch and the same PC in the final epoch. In Dev-AE models, terracing occurs, because along with the growing latent space, an increasing amount of PCs is measured. In both the AE and Dev-AE models, leading PCs are learned early, and remain relatively stable throughout training. In contrast, lower PCs capture little to no variance and therefore show instability.

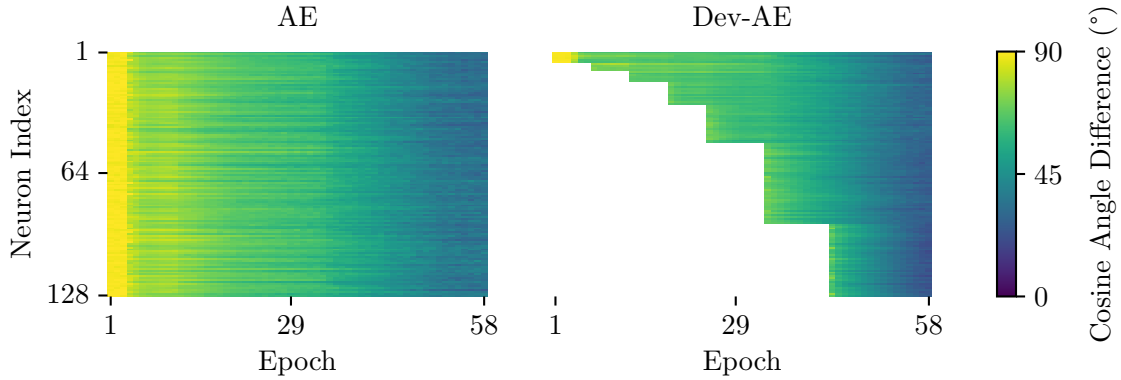
cosine angle difference later during development. Upon closer inspection, some of these differences come in pairs. When the heatmap shows pairs of angle difference, it can be explained by one PC falling below another in the ordering of explained variance. This results in the two principal components switching places in the ordering, but does not necessarily mean that there is a significant change in represented features. The other notable trend is the instability shown in the lowest principal components of MNIST and almost all principal components of CIFAR-10 models. These principal components show no stabilization over time, presumably due to the low intrinsic dimensionality of the dataset. The top principal components justify meaningful variances in the data, while the remaining principal components do not capture any further features, leading to them being redundant.

The Dev-AE follows a similar trend to the AE, with stability coming to leading principal components first before being reached by lower principal components at a later point (when disregarding the redundant principal components). There is, however, a considerable effect of destabilization associated with each increase of the latent space, most apparent in the MNIST figures. In the early stages of training, the introduction of new neurons causes instability across the entire range of principal components. Later increases only affect the lowest of the previously evaluated principal components. The conclusion drawn is that the structuring of the

latent space is primarily affected by early increases, but remains stable in the face of increases occurring later in development.




**(a)** MNIST




**(b)** CIFAR-10

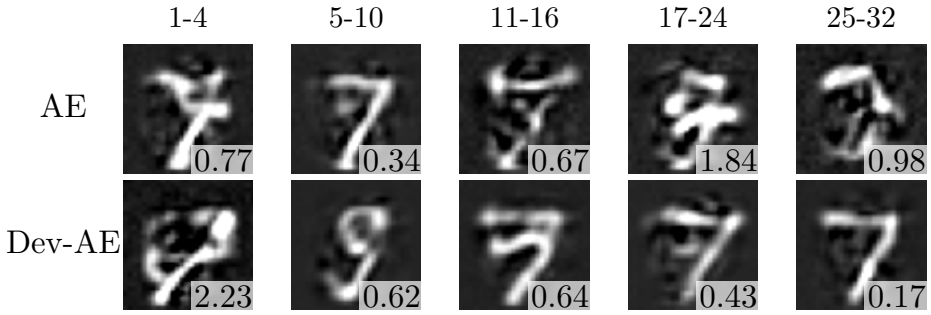
**Figure 16:** Stability of the receptive fields (RFs) of each neuron in the bottleneck layer. Cosine angle difference is measured between a given RF at a specific epoch and the same RF in the final epoch. In Dev-AE models, terracing occurs, because the amount of neurons in the bottleneck is incrementally increased. In both the AE and Dev-AE models, RFs remain relatively stable throughout training, with convergence being reached faster in models trained on MNIST (a) than models trained on CIFAR-10 (b).

The receptive fields of all neurons show stability in both the AE and the Dev-AE. The receptive fields of MNIST stabilize faster. Meanwhile, the CIFAR-10 models show little development at the beginning, as the neural network is still taking initial steps after noisy initialization. Interestingly, this rough patch is overcome quicker by the Dev-AE, possibly because it has fewer neurons in the bottleneck, which then converge faster in learning their features. The Dev-AE's receptive fields are not as sensitive to the latent space increases as is the case with the principal components. In fact, the Dev-AE's neurons stabilize very quickly after the latent space is increased. Examples of the receptive field development of a single neuron are included in the Appendix (see Figure 24), for further visualization of the stability.

In summary, both the principal components and the receptive fields remain relatively stable throughout development. Therefore, the results described in the following subsections concerning the hierarchical ordering, as well as the previous findings concerning disentanglement and learned features (especially frequency), can be generalized to earlier epochs.

### 4.3.2 Importance for Reconstruction

For the sake of assessing the existence of a hierarchy of importance across neuron groups, one may begin by analyzing how the reconstructions of images are affected when the encodings are manipulated for specific neuron groups. The approach for this, detailed in Section 3.7.2, adds noise to a specific neuron group, matches it to this neuron group's mean and standard deviation, and normalizes for group size. This ensures that the neuron groups endure similar perturbations. These noisy encodings are fed into the decoder of the models to obtain a reconstruction. The MSE loss of this reconstruction is inspected, to get an idea of the effects of noise on reconstruction. Example images with noise perturbations are displayed, to provide context as to what the noise affects.

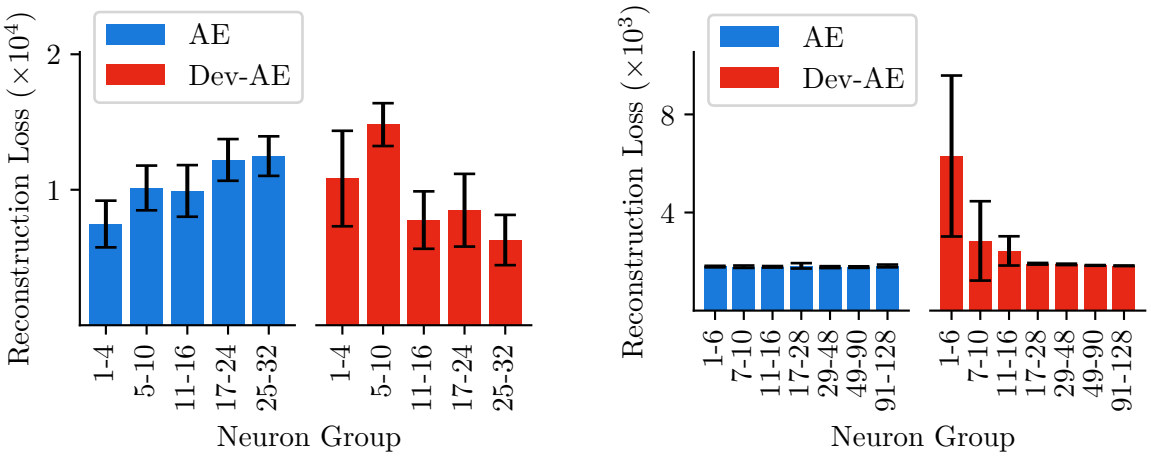



**(a)** MNIST


**(b)** CIFAR-10

**Figure 17:** Examples of a reconstructed image from noise perturbed encodings. Noise is added to each neuron group individually. Each image is labeled with the loss in regards to the original image measured using MSE. The AE models show relatively consistent loss, regardless of the manipulated neuron group, while the Dev-AE models exhibit higher loss for earlier neuron groups.

Figure 17 demonstrates that noise added to early neuron groups in the Dev-AE has a stronger negative impact on reconstruction performance. In the AE of both datasets, the reconstruction performance is very similar for noise in all neuron groups; slight deviations are considered normal when working with noise, this being a specific example. Average quantified results will follow. The effect of noise in the Dev-AE trends downwards, with the earliest neuron group being effected most. Especially in the CIFAR-10 example, the features being targeted are evident. The earliest neuron group affects the reconstruction in a coarse manner, reducing the image to its broad shapes and color. The later Dev-AE neuron groups affect the images more finely, in the same way as happens in the AE across all neuron groups. In order to gain a quantitative insight, this thesis will apply the same approach, this time measured over many models and thousands of test images.




**(a)** MNIST **(b)** CIFAR-10

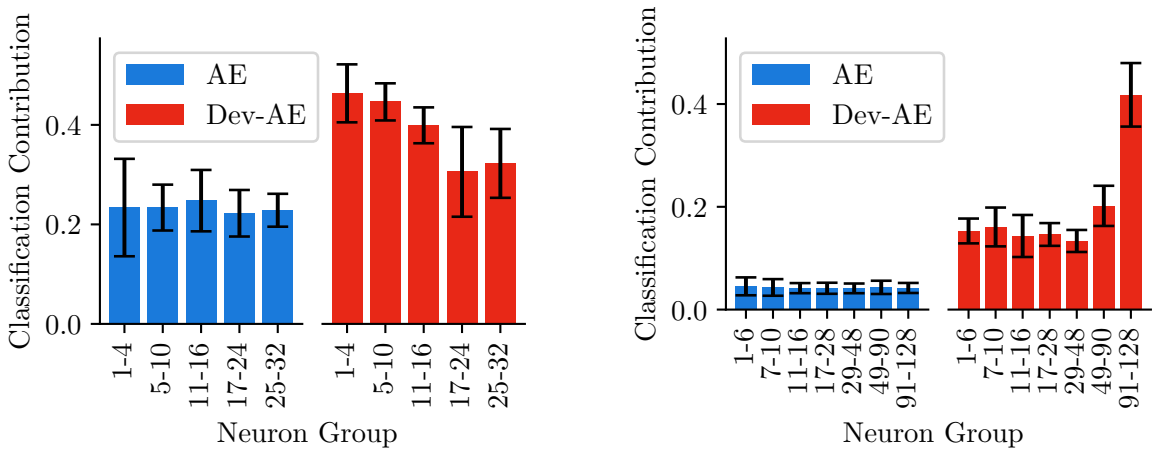
**Figure 18:** Reconstruction loss across neuron groups with noise added to select groups in the encodings. AE models exhibit relatively constant reconstruction loss, regardless of the manipulated neuron group, while the Dev-AE models follow a trend of higher loss for earlier neuron groups.

The results of Figure 18 demonstrate that the results of Figure 17 hold when tested across many images and models, the trend being quite pronounced in both datasets. Especially in CIFAR-10 models, the AE has similar reconstruction loss, regardless of which neuron group is impacted, while, in the Dev-AE, noise added to the earlier neurons results in much higher reconstruction loss than noise on later neurons. For MNIST, the result is not quite as pronounced, but this can be explained by the fact that the dataset is less complex, providing fewer differentiation between neuron groups. Focusing on the results of the more complex CIFAR-10 dataset indicates that early neuron groups in the Dev-AE are the most important for the reconstruction task, with the curved trend being pronounced. Thus, the Dev-AE does indeed show a hierarchy of importance: the earlier a neuron group, the more important it is for the task of reconstruction. A striking difference between the CIFAR-10 AE and Dev-AE models is that the overall reconstruction loss of neuron groups is lower or comparable in every neuron group. This is not mirrored in the MNIST models, but does suggest a trade-off like the one found in $\beta$-VAEs [10]. The gains in interpretability of the latent space are accompanied by a trade-off, where the reconstruction loss of singular neuron groups in the Dev-AE is not as strong as in the AE.

### 4.3.3 Importance for Classification

Measuring reconstruction loss has given valuable insights into the behavior of the whole autoencoder, which entails both the encoder and the decoder. The earlier neurons, which have been proven to learn low-frequency, broad features, were the ones most influential to reducing MSE loss, especially because low frequency relates to the broad features of the image, which when perturbed, cause a higher MSE loss. Next, the neuron groups are examined regarding importance for classification, more specifically the classification of encodings. This approach only uses the encoder. Excluding the decoder allows the focus to be on the latent space. Since the fact that neuron groups learn different frequencies has been established, the relevance of features in an image for its classification can be investigated: either the broader details, or the finer ones. For this measurement, the approach detailed in Section 3.8 is applied: a logistic regression classifier is trained on encodings, then the weight of each neuron for a successful

prediction is assessed. The higher a neuron's measured weight, the more important it is for classification.




**(a)** MNIST **(b)** CIFAR-10

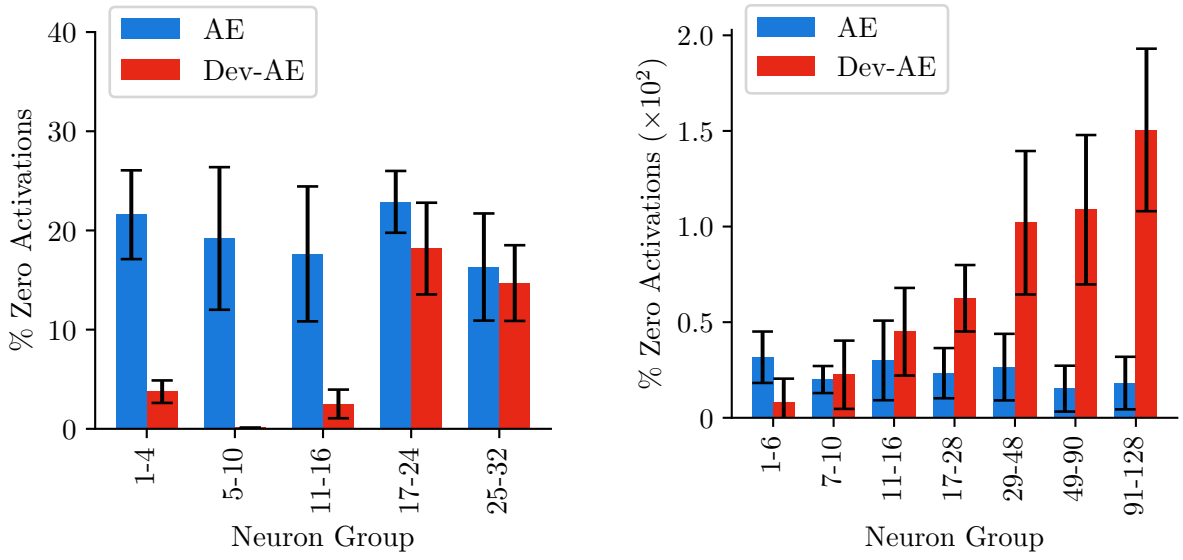
**Figure 19:** Contribution of neuron groups to the classification of an encoding. Classification influence is measured by taking the weights of a logistic regression classifier trained to predict the class of an image given an encoding. Late neuron groups influence classification most in the Dev-AE models trained on CIFAR-10 (b). This trend is not mirrored in the AE or the Dev-AEs trained on MNIST (a).

When regarding Figure 19 it is worth mentioning that the across-the-board lower level of classification influence in the AE has no implications. Unlike the classification accuracy measured earlier (see Figure 14), where the approach allowed for exact comparison, in this figure only the trend is significant. The neuron groups of the AE are, once again, all very similarly important for the classification task. For the Dev-AE, the datasets provide opposing results. In both there is an almost hierarchical ordering: for MNIST, earlier neuron groups are important, and for CIFAR-10 later groups are more important for classification. Again, this difference seems to stem from the complexity of the dataset being analyzed. The finding from CIFAR-10, which would be more likely to be found when testing models on other complex datasets, indicates that, when classifying an image using logistic regression, the most important features are the fine details that correspond to high frequency. This finding stands out, as it shows that the early Dev-AE neuron groups are not always more important. Since the autoencoder learns by comparing its reconstructions with the original image via MSE loss, that is precisely what early neurons optimize for; this is why the early neuron groups of the Dev-AE are so effective at reconstructing broad features. However, in the classification task, it is the later added neurons which are most important for predicting the correct class of an encoding.

## 4.4 Sparsity Analysis of Neural Activations

Measuring sparsity and average activation of neurons in the bottleneck is done using the methods described in Section 3.9. Sparsity measures allow for an evaluation of efficiency. Higher sparsity leads to improved computational efficiency, as zero activations can be skipped during computation. Meanwhile, average activation per neuron provides an interpretation of which neurons are the dominant responders to an image, while average activation per image shows general activation trends. Evaluating the hidden layers with the sparsity method allows for an analysis of whether the effects of an incremental increase are limited to just the latent space, or if its effects in the latent space also propagate to neighboring hidden layers.

### 4.4.1 Sparsity in the Latent Space




**(a)** MNIST **(b)** CIFAR-10

**Figure 20:** Sparsity of neuron groups in the latent space. Measured by counting true zero activations for the models trained on MNIST (a), and by counting near-zero activations for models trained on CIFAR-10 (b). Both show consistency across AE neuron groups and a trend of increasing sparsity with later Dev-AE neuron groups. In MNIST, the AE shows higher sparsity overall, and for CIFAR-10, the Dev-AE shows higher sparsity.

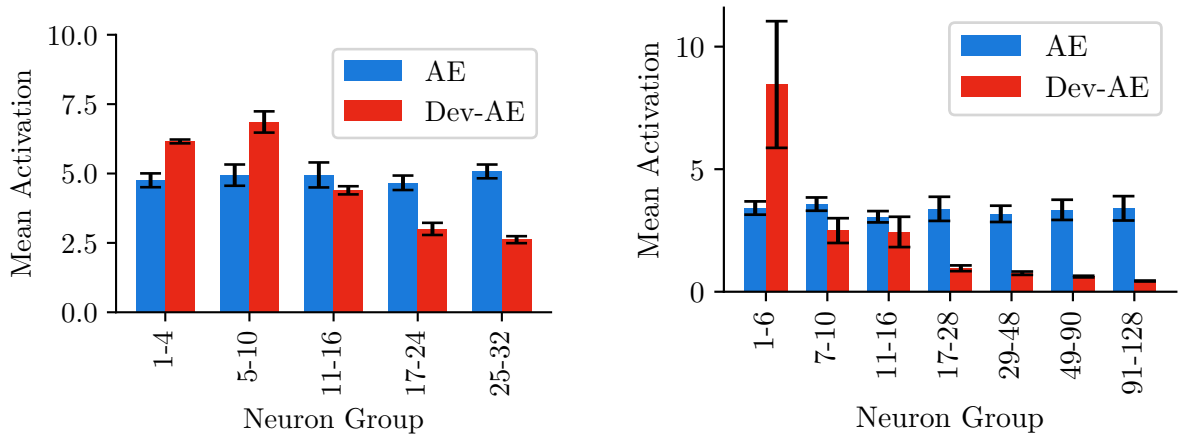
Sparsity in neuron groups shows a clear trend. Regardless of the training dataset, the sparsity of the AE remains relatively constant across all neuron groups. Meanwhile, the Dev-AE shows higher sparsity in later neuron groups. This may be linked to the varying levels of influence that neuron groups have on reconstruction (see Section 4.3.2). Later groups are less important and thus are activated less often. Overall, the bottleneck neurons of the AE trained on MNIST are more sparse than those of the Dev-AE. In contrast, the results of the CIFAR-10 models find that Dev-AE neurons are far more sparse than those of the AE. This is again due to the differences in complexity of the dataset. From this measure alone, it is difficult to conclude whether the Dev-AE creates sparser encodings or not.

Next, the average activation of neuron groups are regarded where zero-activations are disregarded, so as to only examine active neurons. A reverse trend is captured, where the groups that were less sparse show higher average activity. This is in line with the findings that early neuron groups are learning broad features (see Figure 12), which are highly present in all images, thus causing a high activation across all images paired with low sparsity as shown in Figure 20. Instead of averaging by neuron, the sparsity values can also be averaged per image.

For both datasets, the Dev-AE yields lower activations per image. The neurons are firing less overall. The effect is far stronger in the CIFAR-10 models. This coincides with a more constricted latent space during early training, where neurons in the neighboring encoder layer have less influence on the neurons in the bottleneck.

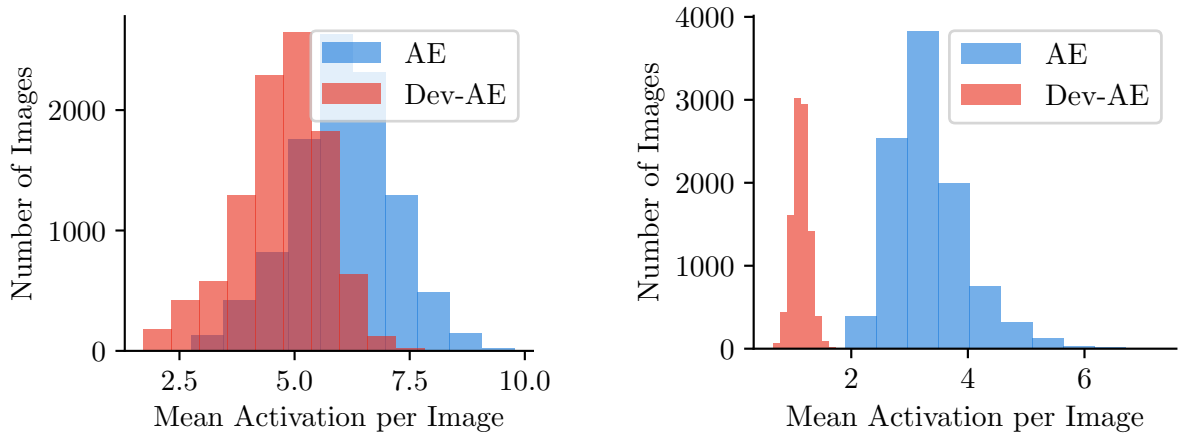
### 4.4.2 Sparsity in the Hidden Layers

Sparsity in the hidden layers measures whether the effects of the developing training extend beyond the latent space itself. Activations are measured over all test images. With this, all neurons are assigned one of three categories. Universally active neurons have a non-zero




**(a)** MNIST **(b)** CIFAR-10

**Figure 21:** Average activation in the bottleneck layer across neuron groups, excluding inactive neurons. AE models exhibit relatively constant activations across all neuron groups, while the Dev-AE models follow a trend of higher activation for earlier neuron groups.



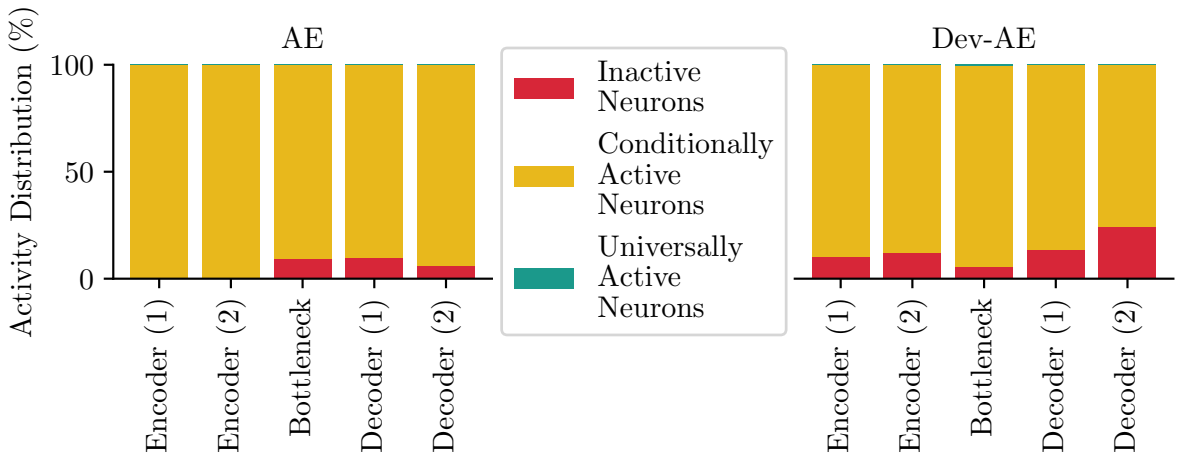

**(a)** MNIST **(b)** CIFAR-10

**Figure 22:** Average activation in the bottleneck layer per image, excluding inactive neurons. The Dev-AE models show lower activation than the AE models. The effect is especially pronounced for Dev-AE models trained on CIFAR-10.

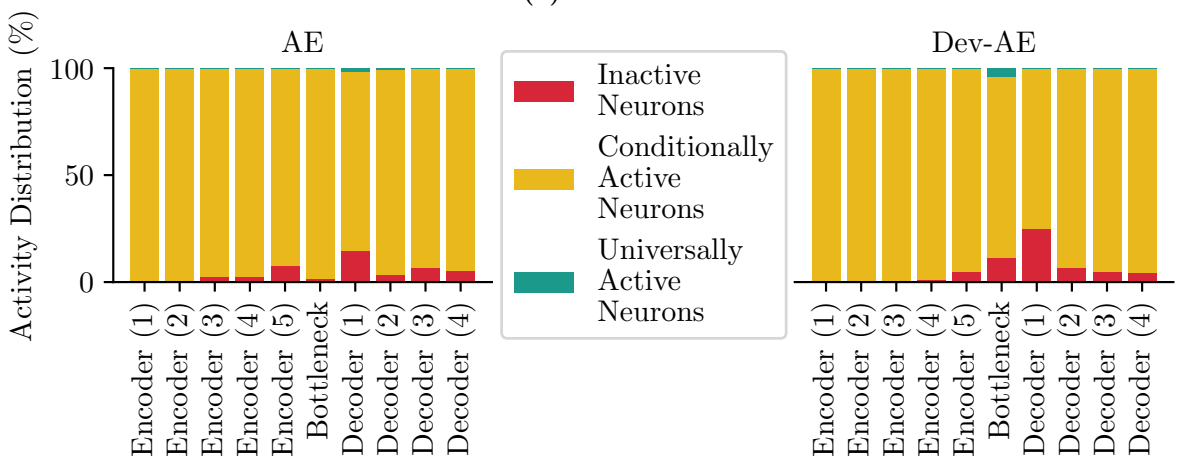
activation for every image fed into the model, implying that they have learned essential features. Conditionally active neurons are active for some of the input images, and the remaining neurons are classed as inactive.

Figure 23 shows that sparsity in hidden layers is indeed affected by the developing training manner. Higher sparsity in hidden layers is observed for models trained on both datasets. The effect is striking for MNIST where sparsity of regular models is not merely amplified, but sparsity appears in layers where it previously was not found. In the CIFAR-10 models, the percentage of universally active neurons increases, which is in line with early neurons learning the most important features, present in all images, which leads to universal activation.

The findings prove that the Dev-AE shows a trend of increasing sparsity for later neuron groups, and lower sparsity in general for the CIFAR-10 models. The measures also found lower activations than corresponding AEs. Previous work [18] has used algorithms to specifically tune




**(a)** MNIST




**(b)** CIFAR-10

**Figure 23:** Activation of neurons in all hidden layers. Images are passed through the model and zero activations are measured. Neurons that are always zero are classed as inactive neurons, sometimes zero as conditionally active neurons, and never zero are called universally active neurons. Sparsity is higher in the Dev-AE models, paired with a slight increase of universally active neurons, in comparison to the AE models.

sparsity, a result that the Dev-AE produces as a byproduct of its developing training manner. The sparsity in hidden layers also proves that the effects of the latent space increase extend beyond the bottleneck and yield higher sparsity in all layers.

# 5 Discussion

## 5.1 Interpretation of Key Findings

Inspired by the increasing dimensionality in the brain, the previous research [2] introduced the novel Dev-AE, a model in which the latent space is incrementally increased during development. To analyze the effects of the developing training method, existing AEs are used as control. Models are trained on the MNIST and CIFAR-10 datasets. Results from prior Dev-AE implementations ([2], [7]) are reproduced and show that Dev-AEs converge to the same loss as the AEs, albeit requiring longer training times. Comparing these fully trained models, with equivalent performance, provides the foundation for the analysis of effects of an incremental increase in the latent space.

Although CIFAR-10 contains natural images, both datasets are relatively simple, revealed by the low intrinsic dimensionality calculated from the autoencoder's latent space. Even still, the natural images of CIFAR-10 showcase a wider range of frequencies. This allows for a disentanglement of frequencies in the models trained on these images, which was not present in the models trained on MNIST. Due to this, the results of the CIFAR-10 models provide more valuable insights with respect to natural images and are given greater emphasis in the evaluation. The analysis of these two datasets suggests that the findings would hold for other datasets, especially more complex ones, which would provide more diverse features to learn and reduce.

The overarching effect of the incremental increase is that neurons learn to represent as many features as the capacity allows. As the capacity grows, so does the number of represented features. Existing latent space neurons were found to maintain their learned representations as new neurons were added. Meanwhile, the new neurons capture the next most important features that were not previously represented. Parallels can be drawn to the visual cortex, in that initial neurons respond to coarse features, while later ones specialize in fine details [5]. However, while each biological neuron specializes over time, the Dev-AE refines its representations through the introduction of new neurons. Each neuron group is similar to one point in time in the development of the brain. Early neuron groups respond to features that early biological neurons would respond to, while later groups resemble the function of more mature neurons.

This incremental feature buildup in the Dev-AE yields disentanglement between neuron groups. Early neuron groups learn low-frequency features and capture the data's leading principal components. The resulting separation improves generalization, a common advantage of disentanglement, when compared to the AE, demonstrated by the comparison of classification accuracy. Furthermore, the existence of distinct representations improves interpretability. It establishes that select neuron groups excel at specific downstream tasks.

While reconstruction depends more heavily on earlier neuron groups, classification is more reliant on later neurons. Thus there is a hierarchical ordering of importance that varies by task. In the CIFAR-10 models, these trends are monotonic, either increasing or decreasing in importance across neuron groups. Such structure improves interpretability, making it easier to determine which neuron groups are most influential for specific tasks. Additionally, the two-way trend highlights the necessity of the later neurons, which appear to be less important in many measures. Since early neurons learn features that most reduce reconstruction loss, which is the most important task during training, they dominate in related tests. The distinct role of the later neurons emphasizes that, to train an autoencoder, which can perform well on downstream tasks, it requires more than just the early neurons.

Earlier neurons were also found to be less sparse than late neurons, highlighting their capturing of broad features that exist in all data points. The sparsity and the lower overall activations are perhaps a remnant of the very small bottleneck during early development, supported by the finding that the initially strongly limited CIFAR-10 latent space showed far lower average activations per image. Sparsity was found to extend beyond the latent space to other hidden layers in the model, verifying that the effects of the incremental increase impact more than just the latent space itself.

The Dev-AE's approach is a dynamic one, similar to approaches discussed in Section 2.2. These dynamic approaches aim to adapt the model's architecture to the task at hand to optimize performance and efficiency, often relying on additional tuning parameters or algorithms to guide the approach. In contrast, the biologically inspired Dev-AE approach is more hands-off. Its improvements in disentanglement, interpretability, and sparsity are inherent to the developing training method. Similar to the previous approaches, the findings support that small, especially biologically motivated changes to model architecture can yield significant beneficial results. The results also show the importance of learning from nature when trying to design effective learning models.

### 5.2 Limitations and Future Work

Despite the Dev-AE model being inspired by the increasing dimensionality in the neural cortices [5], it does not capture the linked concept of correlation or any other causes driving the brain's dimensionality growth. While this abstraction is suitable to find autoencoder training schemes in a machine learning context, it does not model any other complexities of brain development. Future work could address this by exploring a model with stronger biological foundations. One feature that could tie the model closer to biology could be to force existing latent space neurons to split and specialize. This would model the development of neurons, instead of capturing fixed points in time as done in the current approach.

Another direction for future research would be to experiment with different latent space increases. This thesis was limited to introducing numbers of neurons in groups that demonstrated good reconstruction performance. Investigating two extremes could provide interesting results. Increasing the latent space by a single neuron at a time could promote disentanglement, while increasing by doubling would lead to all neurons being initialized as noisy clones of one another, as defined by the initialization scheme of new neurons.

Practical applications could be found when looking further into the concept of dynamic growth. The adapted Dev-AE model presented in this thesis already tracks reconstruction loss during training, to determine when to increase. Future work could look into dynamically finding the number of new neurons introduced at a time, perhaps in relation to how much of the intrinsic dimensionality remains to be captured. In addition to this, hidden layers could also be expanded during development or entirely new hidden layers could be introduced, linking to previous work [14].

## 6 Conclusion

This thesis adapted the existing Dev-AE model and used a variety of methods to better understand the effects of the incremental increase of the latent space. The findings showed improvements in disentanglement, sparsity, and hierarchical feature organization, when compared to architecturally equivalent AE models. These traits are inherent to the developing training manner and required no additional tuning. The understanding of the models gained in this thesis can be used in practical applications, where the newly found benefits of the Dev-AE are required. The work aligns with previous work in dynamic neural networks, and provides directions towards an even more dynamic network, based on the obtained results.

## References


[1] S. Walczak and N. Cerpa. "Artificial Neural Networks". In: *Encyclopedia of Physical Science and Technology (Third Edition)*. Ed. by Robert A. Meyers. Third Edition. New York: Academic Press, 2003, pp. 631–645. ISBN: 978-0-12-227410-7. DOI: 10.1016/B0-12-227410-5/00837-1.

[2] M. Genios. "Representation Learning with Increasing Dimensionality in Autoencoders". Unpublished. Bachelor's Thesis. Goethe University Frankfurt, 2024.

[3] A. Chandna. "Natural history of the development of visual acuity in infants". In: *Eye (London)* 5.Pt 1 (1991). PMID: 2060665, pp. 20–26. DOI: 10.1038/eye.1991.4.

[4] R. Pang, B. J. Lansdell, and A. L. Fairhall. "Dimensionality reduction in neuroscience". In: *Current Biology* 26.14 (2016), R656–R660. ISSN: 0960-9822. DOI: 10.1016/j.cub.2016.05.029.

[5] N. J. Powell, B. Hein, D. Kong, J. Elpelt, H. N. Mulholland, R. A. Holland, M. Kaschube, and G. B. Smith. "Developmental maturation of millimeter-scale functional networks across brain areas". In: *Cerebral Cortex* 35.2 (2025). ISSN: 1460-2199. DOI: 10.1093/cercor/bhaf007.

[6] S. Graven and J. Browne. "Auditory Development in the Fetus and Infant". In: *Newborn and Infant Nursing Reviews* 8 (2008), pp. 187–193. DOI: 10.1053/j.nainr.2008.10.010.

[7] D. Kong, M. Genios, G. B. Smith, and M. Kaschube. "Low-dimensional sensory representations early in development facilitate receptive field formation". In: *Bernstein Conference*. Frankfurt am Main, Germany, 2024. DOI: 10.12751/nncn.bc2024.094.

[8] J. B. Levitt. *Receptive Field*. Encyclopedia Britannica. 2018. URL: https://www.britannica.com/science/receptive-field (visited on 04/02/2025).

[9] S. G. Solomon. "Chapter 3 - Retinal ganglion cells and the magnocellular, parvocellular, and koniocellular subcortical visual pathways from the eye to the brain". In: *Neurology of Vision and Visual Disorders*. Ed. by J. J. S. Barton and A. Leff. Vol. 178. Handbook of Clinical Neurology. Elsevier, 2021, pp. 31–50. DOI: 10.1016/B978-0-12-821377-3.00018-0.

[10] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "$\beta$-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=Sy2fzU9gl.

[11] M. N. Hebart, C. Y. Zheng, F. Pereira, and C. I. Baker. "Revealing the multidimensional mental representations of natural objects underlying human similarity judgements". In: *Nature Human Behaviour* 4.11 (2020), pp. 1173–1185. DOI: 10.1038/s41562-020-00951-3.

[12] W. E. Vinje and J. L. Gallant. "Sparse Coding and Decorrelation in Primary Visual Cortex During Natural Vision". In: *Science* 287.5456 (2000), pp. 1273–1276. DOI: 10.1126/science.287.5456.1273.

[13] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu. "Autoencoders and their Applications in Machine Learning: A Survey". In: *Artificial Intelligence Review* 57.2 (2024), p. 28. DOI: 10.1007/s10462-023-10662-6.

[14] R. Mitchell, R. Menzenbach, K. Kersting, and M. Mundt. "Self Expanding Neural Networks". In: *Computing Research Repository eprint Journal (CoRR)* (2023). DOI: 10.48550/arXiv.2307.04526.

[15] B. Appolinary, A. Deaconu, S. Yang, and Q. Li. *Self Expanding Convolutional Neural Networks*. 2024. DOI: 10.48550/arXiv.2401.05686.

[16] G. Sejnova, M. Vavrecka, and K. Stepanova. "Adaptive Compression of the Latent Space in Variational Autoencoders". In: *Artificial Neural Networks and Machine Learning - ICANN 2024*. Springer Nature Switzerland, 2024, pp. 89–101. ISBN: 9783031723322. DOI: 10.1007/978-3-031-72332-2_7.

[17] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner. *Understanding disentangling in β-VAE*. 2018. DOI: 10.48550/arXiv.1804.03599.

[18] A. Makhzani and B. Frey. *k-Sparse Autoencoders*. 2014. DOI: 10.48550/arXiv.1312.5663.

[19] N. Powell, B. Hein, D. Kong, et al. "Universality of modular correlated networks across the developing neocortex". In: *COSYNE2022*. Lisbon, Portugal, 2022. URL: https://www.world-wide.org/cosyne-22/universality-modular-correlated-networks-5a1134a0/.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[21] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647.

[22] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Technical Report 0. Toronto, Ontario: University of Toronto, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[23] D. Kong, D. Vogenauer, J. Elpelt, and M. Kaschube. "Developing Autoencoder: Incremental Bottleneck Expansion Leads to an Informed Latent Space". Submitted to ICANN 2025, under review. 2025.

[24] A. LeNail. "NN-SVG: Publication-Ready Neural Network Architecture Schematics". In: *Journal of Open Source Software* 4.33 (2019), p. 747. DOI: 10.21105/joss.00747.

[25] E. Facco, M. d'Errico, A. Rodriguez, and A. Laio. "Estimating the intrinsic dimension of datasets by a minimal neighborhood information". In: *Scientific Reports* 7 (2017), p. 12140. DOI: 10.1038/s41598-017-11873-y.

[26] K. Pearson. "On Lines and Planes of Closest Fit to Systems of Points in Space". In: *Philosophical Magazine* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[27] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. *Understanding Neural Networks Through Deep Visualization*. 2015. DOI: 10.48550/arXiv.1506.06579.

# A Model Architectures

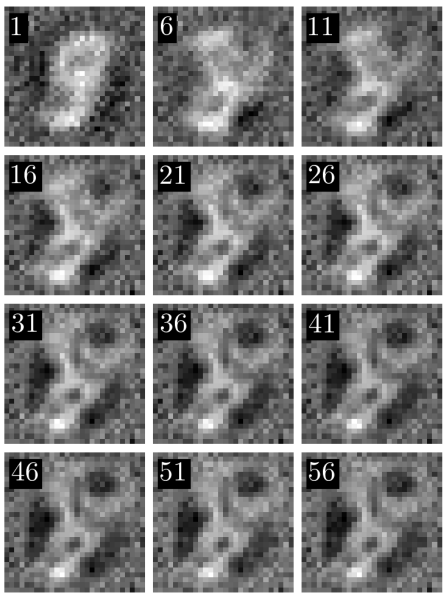
**Table 1:** The MLP autoencoder architecture based on [7]. `latent_dim` remains fixed for the AE and increases during training for the Dev-AE.

| Layer Type | Parameters | Activation | Input Shape | Output Shape |
|---|---|---|---|---|
| | | **Encoder** | | |
| Linear | 784→512 | ReLU | (784) | (512) |
| Linear | 512→128 | ReLU | (512) | (128) |
| Linear | 128→`latent_dim` | ReLU | (128) | (`latent_dim`) |
| | | **Decoder** | | |
| Linear | `latent_dim`→128 | ReLU | (`latent_dim`) | (128) |
| Linear | 128→512 | ReLU | (128) | (512) |
| Linear | 512→784 | - | (512) | (784) |

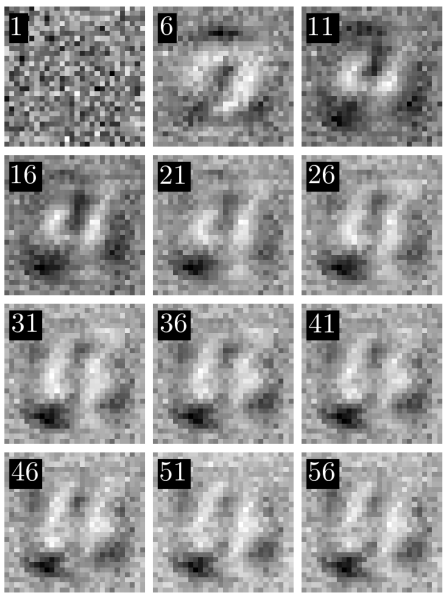
**Table 2:** The CNN autoencoder architecture based on [2]. `latent_dim` remains fixed for the AE and increases during training for the Dev-AE.

| Layer Type | Parameters | Act. | Input | Output |
|---|---|---|---|---|
| | | **Encoder** | | |
| Conv2d | 3→32, kernel 3×3, stride 2, pad 1 | ReLU | (3, 32, 32) | (32, 16, 16) |
| Conv2d | 32→32, kernel 3×3, stride 1, pad 1 | ReLU | (32, 16, 16) | (32, 16, 16) |
| Conv2d | 32→64, kernel 3×3, stride 2, pad 1 | ReLU | (32, 16, 16) | (64, 8, 8) |
| Conv2d | 64→64, kernel 3×3, stride 1, pad 1 | ReLU | (64, 8, 8) | (64, 8, 8) |
| Conv2d | 64→64, kernel 3×3, stride 2, pad 1 | ReLU | (64, 8, 8) | (64, 4, 4) |
| | | **Bottleneck Reshaping** | | |
| Flatten | - | - | (64, 4, 4) | (1024) |
| Linear | 1024→`latent_dim` | - | (1024) | (`latent_dim`) |
| Linear | `latent_dim`→1024 | ReLU | (`latent_dim`) | (1024) |
| | | **Decoder** | | |
| ConvTranspose2d | 64→64, kernel 3×3, stride 2, pad 1, op=1 | ReLU | (64, 4, 4) | (64, 8, 8) |
| Conv2d | 64→64, kernel 3×3, stride 1, pad 1 | ReLU | (64, 8, 8) | (64, 8, 8) |
| ConvTranspose2d | 64→32, kernel 3×3, stride 2, pad 1, op=1 | ReLU | (64, 8, 8) | (32, 16, 16) |
| Conv2d | 32→32, kernel 3×3, stride 1, pad 1 | ReLU | (32, 16, 16) | (32, 16, 16) |
| ConvTranspose2d | 32→3, kernel 3×3, stride 2, pad 1, op=1 | Tanh | (32, 16, 16) | (3, 32, 32) |

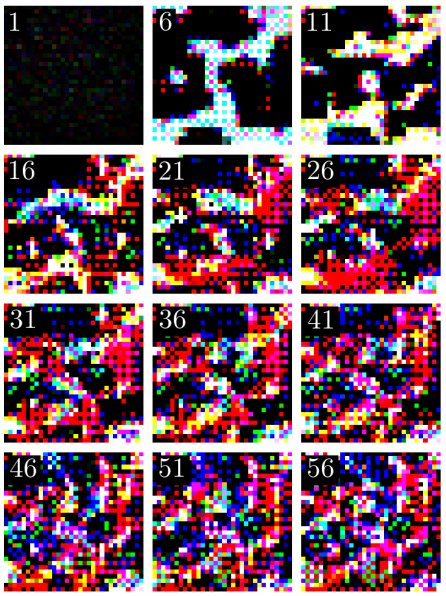
# B Additional Figures



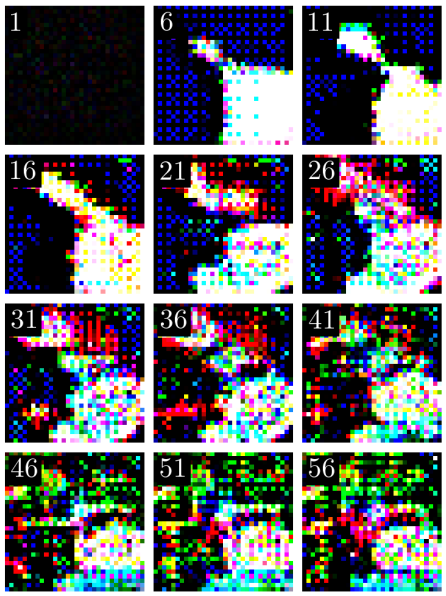

**(a)** AE trained on MNIST




**(b)** Dev-AE trained on MNIST

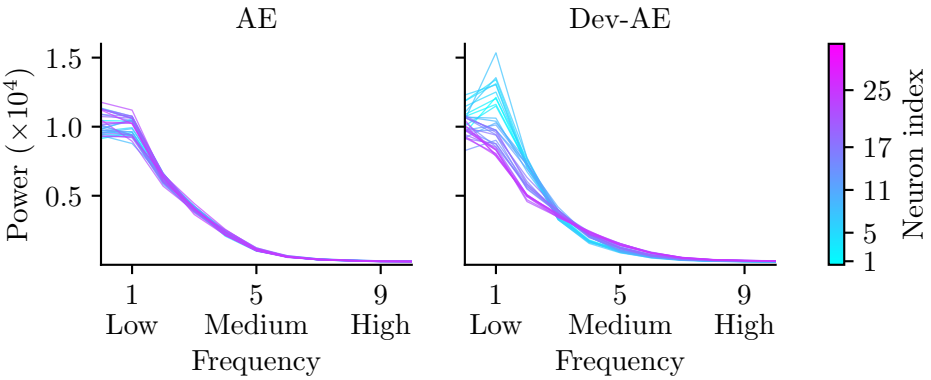



**(c)** AE trained on CIFAR-10

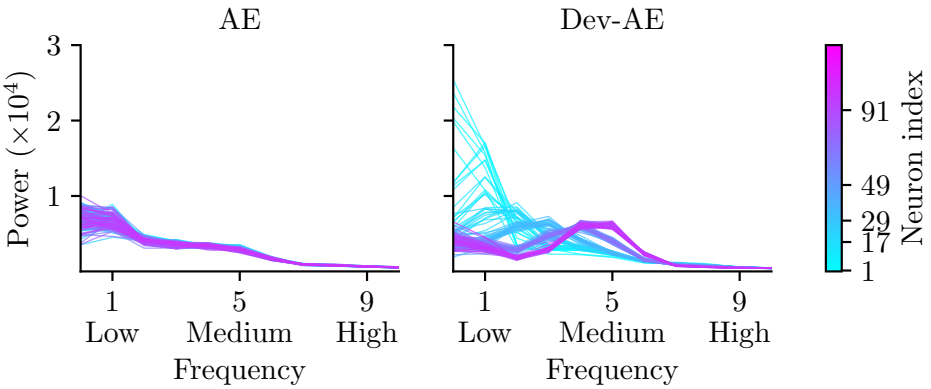



**(d)** Dev-AE trained on CIFAR-10

**Figure 24:** Development of each model's first neuron's receptive field throughout training. An arbitrary model is selected, and the receptive fields after every 5th training epoch are displayed. The label shows after how many epochs each receptive field was computed.

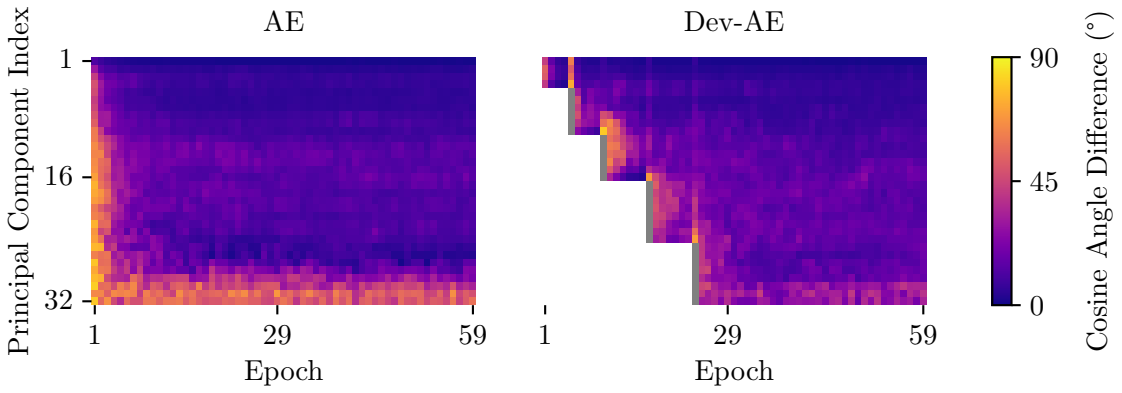



**(a)** MNIST

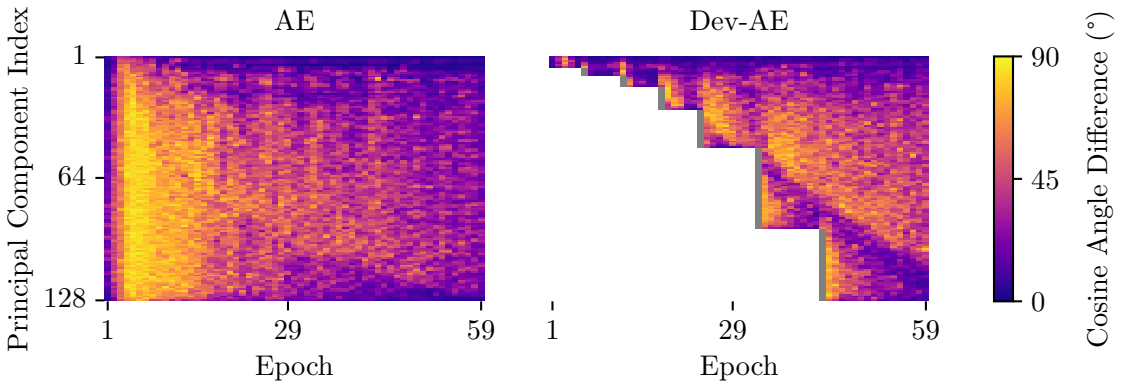



**(b)** CIFAR-10

**Figure 25:** Power spectra of the receptive fields of individual neurons. The results align with those found in Figure 12, except for some separation in between the early neuron groups of the Dev-AE, which was not visible in the grouped plot.

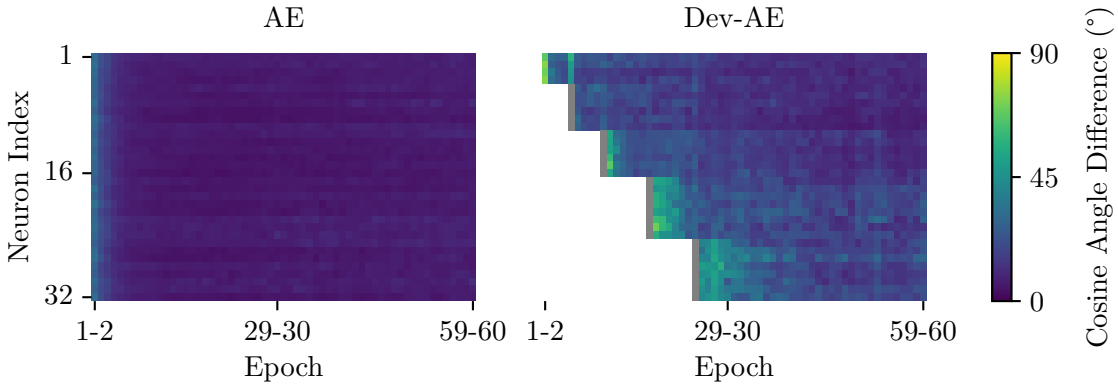



**(a)** MNIST

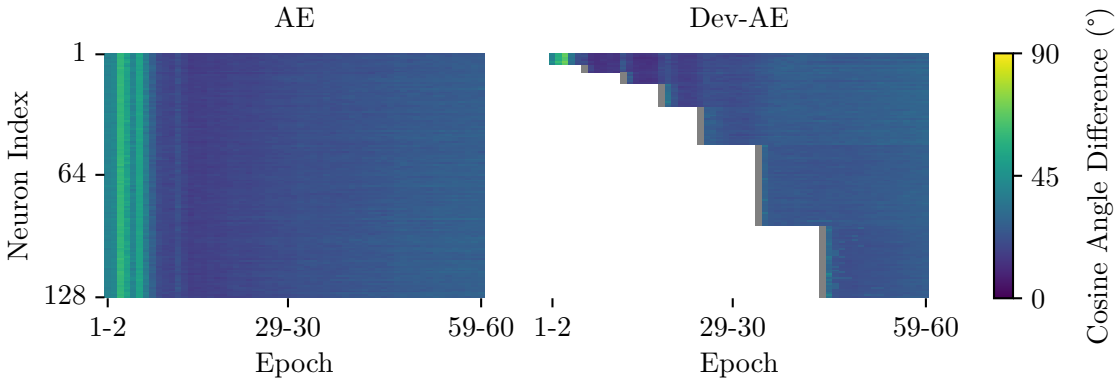



**(b)** CIFAR-10

**Figure 26:** Stability of the principal components in the encodings. Cosine angle difference is measured between consecutive epochs, resulting in similar findings to Figure 15. Gray cells indicate non-computable values, where there is no value in the prior epoch to compare against.




**(a)** MNIST




**(b)** CIFAR-10

**Figure 27:** Stability of the receptive fields of each neuron in the bottleneck layer. Cosine angle difference is measured between consecutive epochs, resulting in similar findings to Figure 16. Gray cells indicate non-computable values, where the neuron in the prior epoch does not yet exist and cannot be compared against.

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

# Erklärung zur Abschlussarbeit

**gemäß § 35, Abs. 16 der Ordnung für den Bachelorstudiengang Informatik vom 17. Juni 2019:**

Hiermit erkläre ich

_____

*(Nachname, Vorname)*

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ich bestätige außerdem, dass die vorliegende Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen meiner vorliegenden Bachelorarbeit mit der digital eingereichten elektronischen Version meiner Bachelorarbeit übereinstimmen.

Frankfurt am Main, den

_____

Unterschrift der/des Studierenden