Master of Science in Engineering with Specialisation in Data Science

Master Thesis

Deep-Learning-based Pattern Recognition and the Principle of Self Organization

Incorporating Findings from Neuroscience about Natural Intelligence into Modern Deep Learning Architectures

Pascal Sager

February 27, 2023

Zurich University of Applied Sciences

"Max Planck said, 'Science progresses one funeral at a time.'
The future depends on some graduate student who is deeply
suspicious of everything I have said."

– Geoffrey Hinton, University of Toronto, 2017.

Zusammenfassung

Abstract

Preface

As soon as I have handed in this thesis and it has been assessed with a sufficient grade, I may use the title "Master of Science". Thus, I should know and have learned something about science. Science can be defined as the systematic analysis of the real or virtual world through observations and experiments as well as the further development of existing technology. While this definition sounds straightforward, it takes a lot of experience and commitment to work successfully in science. I have been lucky enough to be able to learn more about science and to apply my knowledge in research projects at the Centre for Artificial Intelligence (CAI) of the Zurich University of Applied Sciences (ZHAW) since the beginning of my studies. I also have the great privilege of being supported and mentored by Prof. Dr. Thilo Stadelmann, Head of the CAI. He uses to say (also in accordance with his blog-post) "Great methodology delivers great theses". It is always desirable to have an excellent outcome in a thesis such as a system that can execute a task and thereby achieves or even overcomes state-of-the-art performance. However, in my opinion, it is equally or even more important to reason why and how something works, to justify choices, and to show limitations. I wrote my Thesis with these thoughts in mind and hope that the readers can follow my reasoning.

This thesis spans two fields; computer science and neuroscience. I try to link these fields as clearly as possible and to write in a way such that readers from both disciplines can follow my argumentation. However, this also means that some aspects are described rather extensively. So if you as a reader consider yourself a specialist in one of these fields, feel free to skip (parts of) chapters like the "Fundamentals" Chapter. In general, the chapter "Fundamentals" describes principles that are not necessary for understanding this thesis, but are helpful for people with a deep learning background to gain an overview of the field of neurocomputing and vice versa.

I would like to thank a couple of colleagues and friends for their support in this thesis. First I think of my mentor Prof. Dr. Thilo Stadelmann who got me excited about AI years ago and later introduced me to research. He always encouraged creative ideas, thinking outside of the box, and striving for greatness. Thank you for your support, help, and guidance, I have grown personally as well as professionally. Further thanks go to Dr. Jan Deriu. He has always helped to translate abstract ideas into concrete algorithms and to get them running. It's impressive how your understanding of deep learning can make complex problems look so simple. To Prof. Dr. Christoph von der Malsburg for his seemingly endless patience in introducing me to neuroscience. You have inspired me regularly with ideas and opened up a new way of thinking about deep learning (this was also the inspiration for Geoffrey Hinton's quote on the page before, although I wouldn't presume to say that this thesis will change the future). Even though we couldn't implement all of your ideas in this thesis, I learned a lot in our discussions and hope that I will be able to tackle more of your ideas in the future.

The biggest thanks, however, goes to my family, who made this journey possible for me. To my parents, who supported and encouraged me in every way. To my younger brother who inspired me to study. To my wife and son, who have been understanding and supportive and have always been

the perfect counterbalance to the daily routine. Without the support of my family, I would never have been able to embark on this academic path.

Contents

List of Figures

List of Tables

Mathematical Terms & Definitions

1 Notation

Formatting	Example	Meaning
No formatting, lower case	a	A scalar value
Bold, lower case	$\boldsymbol{a}$	A vector
Bold, upper case	$\boldsymbol{A}$	A matrix

2 Data

$\boldsymbol{X}$ Input data that is fed into a model, typically a matrix (since it is a mini-batch with multiple samples)

1 Introduction

Mankind has always tried to simplify its life through technological progress. In the last hundred years, the computer in particular has shaped progress in every field imaginable. The computer has been so successful in fact that it has spawned its own scientific discipline, computer science. Nowadays it is impossible to imagine life without computers: almost every household and company owns computers. Computers facilitate various tasks, from communication to the acquisition of knowledge. For all these tasks, software developers have created appropriate programs. Software development is the process of writing a script with a programming language to specify how the computer should behave for a given input. Simply put: A software program tells the computer what to do if the user enters a command. This works very well if the tasks are clearly defined and can be described precisely. However, there exist tasks that are almost impossible to program such as writing a script that detects cats in images because we cannot describe how a cat looks precisely[1].

1: at least not on a pixel-level basis so that we can simply compare a given image with our description

Therefore, scientists came up with the idea to not just program such tasks but to let the computer learn them instead. Machine learning (ML) are algorithms that can learn and adapt without following explicit instructions such as program code. Instead, they use statistical models to analyze data, find patterns in the data, and draw inferences from it. Machine learning has become an indispensable part of our everyday lives. For example, we use it for machine translation, transport and logistics organization, product recommendations, fraud detection, self-driving cars, unlocking smartphones, improving video games, speech recognition, and much more. A sub-branch of machine learning is deep learning which achieved very impressive results in the last decade and is considered the state-of-the-art technology for many of the aforementioned tasks.

Even though this technology works very well for many tasks, such models have some crucial flaws by definition (c.f. Section Section 2.3) which cannot be resolved for sure in the current DL framework. One of the godfathers of deep learning is the Turing Award winner Geoffrey Hinton. Especially his contribution to backpropagation (c.f. Section Section 2.2) has created the foundation for modern deep learning. More than 30 years later, in 2017, Hinton says that he is "deeply suspicious" about end-to-end backpropagation of error, and in his view we have to "throw it all away and start again" to improve current systems fundamentally [1]. Considering what DL systems have achieved, this seems a bit extreme. However, it also shows that the current learning algorithm of such systems has serious flaws.

[1]: Inc. (2017)

Some of the most crucial limitations of deep learning systems are listed in Section Section 2.3. This thesis addresses some of these problems, especially the robustness problem. This is done by exploring different architectures inspired by findings from the field of neuroscience[2]. The inspiration is drawn from neuroscience as the human brain seems to not have these problems and thus incorporating findings from neuroscience

2: this field studies how the human nervous system and especially the brain works

into the deep learning framework might help to overcome some of the current limitations. In this thesis, the theory of natural intelligence by von der Malsburg et al. [2] in particular is used as an inspirational source. In accordance with the aforementioned theory, visual processing is examined and consequently, the focus is on deep learning architectures for image processing. Furthermore, the main focus of this thesis is on *finding new principles* of deep learning architectures and not on achieving state-of-the-art performance in terms of accuracy or f-score on existing benchmarking datasets.

[2]: Malsburg et al. (2022)

The thesis differs from neurocomputing[3] in the sense that the architectures investigated are still closely related to the DL framework and, for example, do not use time-dynamic signals. For the completeness of this thesis, an overview of neurocomputing is given in Section Section 2.5. However, since these types of algorithms have not (yet) become established in practical applications, they are not investigated in this thesis. Thus, the thesis is strongly oriented along the field of deep learning and focuses on incorporating findings of neuroscience in the deep learning framework while natural plausibility is of less importance.

3: neurocomputing is considered a subfield of neuroscience that deals with the implementation of algorithms with a high degree of plausibility according to neuroscientific findings on how natural nervous systems work

1.1 Contribution

The following contributions are made in this thesis:

- First, the basics of the fields of deep learning and neurocomputing are described in detail. Together with related work, this provides a survey of the related fundamentals.
- Second, promising concepts from the neuroscience literature that may be necessary for more intelligent systems are identified and suggestions are made on how to integrate these concepts into a modern DL framework.
- Third, I present two concrete DL architectures that make use of such neuroscience-inspired concepts and analyze their strengths and weaknesses in detail.
- Fourth, this work gives an impression of which concepts that are currently little used in DL settings seem to be promising and makes a recommendation for future work.

1.2 Organization of Thesis

The remainder of the thesis is organized as follows: In Chapter 2 the fundamentals of deep learning and neurocomputing are explained in detail. Experts may skip this chapter, likewise, the second Section on neurocomputing can optionally be omitted, as it is not fundamental for understanding this thesis. Chapter 3 introduces the related work. Chapter 4 describes the methods used; first, I explain my interpretation on how specific neuroscientific concepts could be implemented in a deep learning setting. Afterwards, two architectures incorporating these concepts are presented. In Chapter 5 the results and findings of this thesis are discussed. Finally, in Chapter 6 future work is described and some

insights and intuitions of the author are given about which neuroscience concepts seem promising and which do not.

Fundamentals 2

Machine learning uses mathematical functions to map an input to an output. These functions usually extract patterns from the input data to build a relationship between input and output. The term machine learning stems from the fact that we use *machines* to correlate the input and the output to a function (i.e. to *learn* a function) during a training period. Deep learning is a sub-branch of machine learning and considered state-of-the-art for many learning tasks, especially on high-dimensional data such as texts, audio recordings, 2D as well as 3D images, and videos. Deep learning algorithms use neural networks that are heavily inspired from connected neurons within the human brain.

Therefore, this chapter first briefly explains how biological neurons work and relates them to artificial neurons (c.f. Section 2.1). Next, Section 2.2 describes artificial neural networks that connect a large number of artificial neurons. In Section 2.3, problems of such artificial neural networks are pointed out. Section 2.4 describes some of the differences of deep learning compared to biological learning and Section 2.5 describes biologically more plausible learning methods that are investigated in the field of neurocomputing. These last two sections are not essential for the understanding of this thesis and can optionally be skipped. However, they are intended as a supplement for interested readers who want to get an overview of the whole field.

2.1 Biological Neurons

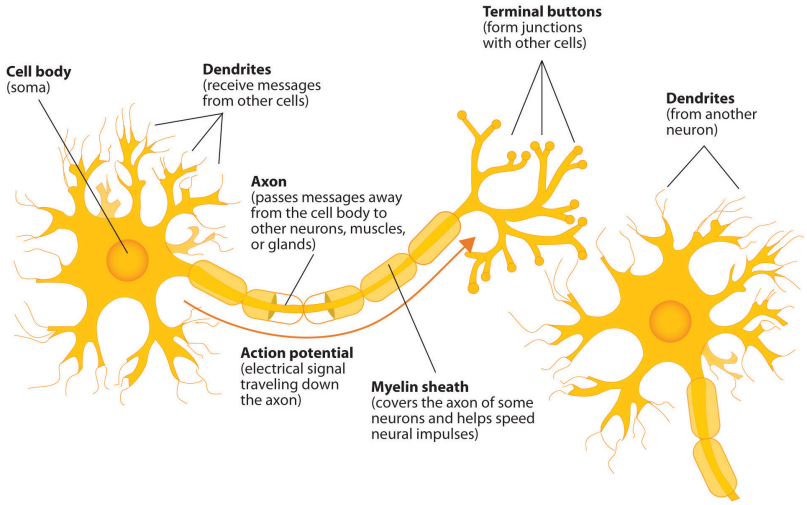

Figure 2.1: A Diagram of the components of a biological neuron. The image is from Wikipedia [3].

A biological neuron (c.f. Figure 2.1) is a cell that communicates with other neurons through connections called synapses. Communication takes place through precisely timed electrical pulses. Biological neurons are electrically excitable by voltage changes across their membranes. If

the changes is large enough within a short interval, the neuron generates a pulse called an action potential. This action potential travels through the axon and activates synaptic connections. Other neurons, connected through synapses, receive this signal. The synaptic signal can be be excitatory [4] or inhibitory [5], making the post-synaptic neuron more or less likely to fire an action potential. Biological neurons are typically classified into three types; sensory neurons, motor neurons, and interneurons. Sensory neurons respond to external stimuli such as light or sound and send their signal to the spinal cord or directly to the brain. Motor neurons receive signals from the brain and spinal cord to control muscles or organs. Interneurons connect neurons within the same region of the brain or the spinal coord. Multiple connected neurons from a neural circuit. The neural network in the brain is not static but changes through growth and reorganization. This process is referred to as neuroplasticity or neural plasticity [6].

[4]: Takagi (2000)

[5]: Coombs et al. (1955)

[6]: Costandi (2016)

An artificial neuron is, similar to a biological neuron, connected to other neurons. Artificial neurons are usually organized in layers that forward signals sequentially. Although the neurons in the first layer could be considered sensory neurons, the neurons in the last layer could be considered motor neurons, and the neurons in the middle layer could be considered interneurons, such a distinction makes less sense because the artificial neurons function similarly regardless of their layer except for the activation function. Several variants for artificial neurons have been proposed in the literature. These variants are described in the following Section 2.2. Similar to biological neurons, multiple artificial neurons are usually connected to an artificial neural network.

2.2 Artificial Neural Networks

The idea for artificial neural networks (ANN) stems from biology and aims to capture the interaction of brain cells (neurons) with a mathematical model. A first model for a neuron was proposed by MuCulloch and Pitts in 1943 [7]. Similar to how a neuron of the human brain transmits electrical impulses through the nervous system, the artificial neuron of MuCulloch and Pitts receives multiple input signals and transforms them into a output signal. A neuron takes an input vector $\boldsymbol{x} = (x_1, ..., x_n)$ where $x_i \in \{0, 1\}$ and maps it to an output $\hat{y} \in \{0, 1\}$. The mapping from the input to the output is done by using an aggregation function g that sums up the input vector $\boldsymbol{x}$ and a activation function f that outputs 1 if the output of g is greater than a threshold θ and 0 otherwise.

[7]: McCulloch et al. (1943)

$$g(\boldsymbol{x}) = g(x_1, ..., x_n) = \sum_{i=1}^{n} x_i \tag{2.1}$$

$$\hat{y} = f(g(\boldsymbol{x})) = \begin{cases} 1, & \text{if } g(x) \geq \theta \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

In 1958, Rosenblatt Rosenblatt [8] developed the Perceptron which works with real numbers as input. The input vector $\boldsymbol{x} = (x_1, ..., x_n)$ where $x_i \in \mathbb{R}^n$ is multiplied with a weight vector $\boldsymbol{w} = (w_1, ..., w_n)$ where $w_i \in \mathbb{R}^n$ with the same length.

[8]: Rosenblatt (1958)

$$g(\boldsymbol{x}) = g(x_1, ..., x_n) = \sum_{i=1}^{n} w_i \cdot x_i \tag{2.3}$$

The output $\hat{y} \in \{0, 1\}$ is similar to the MuCulloch and Pitts neuron 1 if the aggregated value is greater than a threshold θ and 0 otherwise as described in equation **??**. The equations (**??**) and (**??**) can be rewritten as

$$\hat{y} = f(g(\boldsymbol{x})) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Later, the step-function f was replaced with other functions so that the output could also be a real number $\hat{y} \in \mathbb{R}$. Often used functions are

$$\begin{aligned} \text{Sigmoid: } \sigma(z) &= \frac{1}{1 + \mathrm{e}^{-z}} \\ \text{Rectified linear unit (ReLU): } (z)^+ &= \max 0, z \\ \text{Hyperbolic tangent (tanh): } \tanh(z) &= \frac{\mathrm{e}^{z} - \mathrm{e}^{-z}}{\mathrm{e}^{z} + \mathrm{e}^{-z}} \end{aligned} \tag{2.5}$$

By convention, instead of using the threshold $-\theta$ often a bias b is used which leads to:

$$\begin{aligned} z = g(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} &= \sum_{i=1}^{n} w_i \cdot x_i + b \\ \hat{y} &= f(z) \end{aligned} \tag{2.6}$$

However, the brain consists of multiple neurons which are connected through synapses. Therefore, ANNs consist not only of one neuron but combine multiple neurons in a network. These neurons are organized into layers. A shallow neural network consists of one hidden layer in which the input $\boldsymbol{x}$ is fed to calculate the output $\hat{\boldsymbol{y}}$[1]. The universal approximation theorem of Cybenko [9] proves that a shallow network with enough neurons can approximate any mapping function between inputs and outputs. However, very complex mapping functions may need too many hidden neurons. The neurons of the hidden layer extract features in the input space. Since only one layer is used, the features cannot be hierarchically organized and become complex enough.

1: $\hat{\boldsymbol{y}}$ has become a vector since multiple neurons produce multiple output values

[9]: Cybenko (1989)

A multi-layer perceptron (MLP), on the other hand, consists of multiple layers. The different layers extract increasingly complex features. In a MLP. the input $\boldsymbol{x}$ is fed into the first layer, each subsequent layer l gets the output of the previous layer $l - 1$ as input. In the following, we describe the mathematical model for fully connected layers, where all neurons of a layer are connected with the subsequent layer[2]. For a MLP with m layers, we define the output of the aggregation g as $\boldsymbol{z}^{[l]}$ and the output the activation function as $\boldsymbol{a}^{[l]}$ for layer l. Furthermore, we use $\boldsymbol{w}^{[l]}$ for the weight vector and $b^{[l]}$ for the bias of layer l. Thus, the mathematical model of a MLP is defined as

2: many modern network architectures are not fully connected and can have either missing or recurrent connections

$$\begin{aligned} \boldsymbol{z}^{[l]} &= \boldsymbol{w}^{[l]}\boldsymbol{a}^{[l-1]} + b^{[l]} \\ \boldsymbol{a}^{[l]} &= f(z^{[l]}) \end{aligned} \tag{2.7}$$

Since the input is fed into the first layer and the output is the result from the last layer $\boldsymbol{x} = a^{[0]}$ and $\hat{\boldsymbol{y}} = a^{[m]}$ holds true.

So far, only the forward-pass which is used to calculate the output $\hat{\boldsymbol{y}}$ was discussed. However, the model output $\hat{\boldsymbol{y}}$ will only be close to the target output $\boldsymbol{y}$ if the weights $\boldsymbol{w}^{[l]}$ and biases $b^{[l]}$ are properly defined. These parameters are learned during a training period. The training can take place in a supervised, semi-supervised, self-supervised, unsupervised, or reinforcement learning based manner. In supervised learning, the output of the model $\hat{\boldsymbol{y}}$ for a given input $\boldsymbol{x}$ is compared to manually created target outputs $\boldsymbol{y}$. Unsupervised learning, on the other hand, tries to find patterns in the input $\boldsymbol{x}$ and to cluster the samples into meaningful groups without using target labels. Semi-supervised learning is a hybrid approach of the the aforementioned principles that combines a small amount of labelled data with a large amount of unlabelled data. In self-supervised learning, the target outputs $\boldsymbol{y}$ are directly derived from the input data $\boldsymbol{x}$ (e.g. predict a masked part of the input $\boldsymbol{x}$). Lastly, reinforcement learning algorithms aim to maximize a reward that they become from an environment based on some action they executed.

These learning principle have in common that a loss function $\mathcal{L}$ can calculate a loss value based on the model output $\hat{\boldsymbol{y}}$. For example, the mean square error (MSE) can be used for regression problems or the negative log-likelihood for classification problems. The chosen loss function is minimized interatively with stochastic gradient descent (SGD)[3] until the network converges. The idea behind stochastic gradient descent is to make use of the fact that the negative gradient of the loss value points to the direction of the steepest descent (i.e. in the direction where the loss gets smaller). SGD therefore updates the network parameters by taking a step of size η in the direction of their negative gradient

3: There exist also other optimizer methods such as SGD with momentum, RMSprop, or Adam [10]

$$\begin{aligned} \Delta\boldsymbol{w}^{[l]} &= -\eta\nabla_{\boldsymbol{w}^{[l]}}\mathcal{L} \\ \Delta b^{[l]} &= -\eta\nabla_{b^{[l]}}\mathcal{L} \end{aligned} \tag{2.8}$$

The gradients of the weights $\boldsymbol{w}^{[l]}$ and biases $b^{[l]}$ can efficiently be calculated with an algorithm called backpropagation [11], which is just a smart implementation of the chain rule[4].

[11]: Rumelhart et al. (1986)

4: While a detailed discussion on backpropagation is out of scope for this thesis, we refer interested reader to the deep learning course by Andrew Ng [12]

One of the most important design decisions in creating ANNs is how the neurons are connected. If the each neuron is connected to each neuron of the succeeding layer, it is called a Fully Connected Layer (FCN). There exists several alternatives besides such dense connections. Since this thesis deals with the processing of visual scenes, only the most important neural network architectures for vision applications are presented in the following.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are particularly useful for finding patterns in images but can also be used to analyze non-image data such as audio data or time series. Similar to FCNs, a CNN is composed of an input layer, an output layer, and many hidden layers in between. A typical CNN consists of subsequently connected convolutional layers and pooling layers. Usually, an activation function is applied after each convolutional layer while no activation function is used after pooling layers. Depending on the task, the last layers can be different, e.g. for image classification the last layers are often fully connected.

Convolutional layers use convolution filters or kernels that slide along input features and create translation-equivariant[5] responses known as feature maps [14]. The size of the filter, which is typically a 3×3 matrix, determines the size of the receptive field. The filter is applied to an area of the input, with that area having the same size as the filter. When applying the filter, the dot product is calculated between the input and the filter and then fed into an output array (i.e. the feature map). Afterwards, the filter shifts by a stride, repeating the process until the entire input has been processed. Since only the kernel have to be learned, convolutional layers consists of much less parameters than equivalently sized fully connected layers. This process of re-using the same weights at different locations of the input is also known as parameter sharing. As described earlier, multiple convolutional layer can follow on each other. By doing so, CNNs become hierarchical as the later layers can see the pixels within the receptive fields of prior layers.

5: most CNNs apply downsampling operations to the input and are therefore not translation-invariant but translation-equivariant [13]

[14]: Zhang et al. (1990)

Pooling layers reduce the size of the input by conducting dimensionality reduction. Similar to convolutional layers, a filter slides along the input. However, this filter does not have any learned parameter but apply an aggregation function. Usually, the filter either select the pixel with the highest value (max pooling) or calculates the average (average pooling) within the receptive field and forwards it to the output array. Pooling layers usually discard a lot of information but help to reduce complexity and increase robustness.

In the last decades, various CNN architectures have been proposed, usually consisting of different combinations of CNN and pooling layer. Further improvements have been achieved by using parallel paths of convolutional layers, batch normalization [15], and skip connections[6]. In this thesis I do not go into these specific architectures but provide some references to some well known architectures; LeNet [16], AlexNet [17], VGGNet [18], GoogLeNet [19], ResNet [20], U-Net [21], Mask R-CNN [22], SSD [23], and YOLO [24].

[15]: Ioffe et al. (2015)

6: skip connections skips some of the layers in the network and add the output of a layer to the input of a later layer

[16]: Lecun et al. (1998)

[17]: Krizhevsky et al. (2012)

[18]: Simonyan et al. (2015)

[19]: Szegedy et al. (2014)

[20]: He et al. (2016)

[21]: Ronneberger et al. (2015)

[22]: He et al. (2017)

[23]: Liu et al. (2016)

[24]: Redmon et al. (2016)

Training of Convolutional Neural Networks

CNNs can be used in various applications such as image recognition, video analysis, natural language processing, anomaly detection in time series and many others. Since this thesis deals with better representations for visual scenes, this chapter is limited to the typical image analysis tasks; image classification, object detection, and image segmentation. An overview of these tasks is shown in Figure 2.2. The goal of image

classification is to predict an image-level label, i.e. to predict what object is in the image. A classic example of this problem is predicting whether a cat or a dog is in a picture. This is typically done for images that only contain one object. With a multi-label classifier, however, it is also possible to predict whether none, one or several classes are present, i.e. whether a cat, a dog or both are present in the image. With image classification it is unclear where in the image these objects are located. Object detection provides a remedy. The aim of object detection is not only to determine what is visible in the image but also where. The position of the object is usually indicated by a bounding box (i.e. a rectangle). Especially when there are several objects within a picture, it is helpful to know which object is where, e.g. that the cat is to the left of the dog. For some applications, predicting the position with bounding boxes is not sufficient. In this case, semantic segmentation is often used. Semantic segmentation is the task where each pixel is classified (the image is divided into segments) which leads to a pixel-wise mask for each object in the image. This gives us exact information about the shapes of the objects.

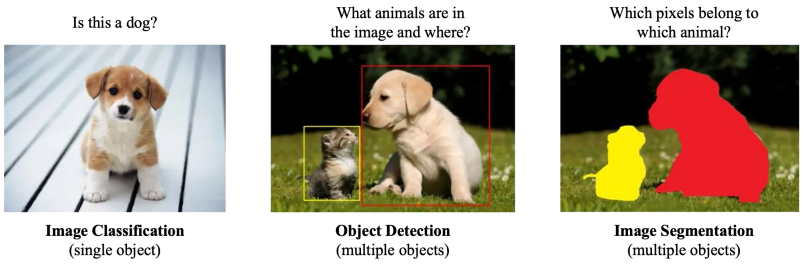

Figure 2.2: Overview of different image analysis tasks. The image is taken from Beat [25] and was slightly adapted.

Depending on the task, different kind of labels are required for supervised learning. Image classification requires labels on image-level (i.e. one or multiple labels per image) [16–20], object detection requires the coordinates of the bounding box [22–24], and semantic segmentation requires the labels on pixel-level [21, 26] (i.e. each pixel has a label of an object or a label "background" for irrelevant pixels).

Besides supervised learning, there are various principles of learning with partial labels or without labels. These techniques are called weakly-supervised or un-supervised learning and are well summarised in [27]. Not only specific tasks can be learned, but also task-independent representations. These representations are typically learned unsupervised[7] and can be used for one or several downstream tasks. More details on visual representation learning are provided in Section 3.4. Especially Autoencoders [28] are explained in this section as they are applied in this thesis.

[27]: Simmler et al. (2021)

7: also called self-supervised learning because the target labels are derived from the data itself

[28]: Rumelhart et al. (1985)

2.3 Limitations of Deep Learning

The rise of deep learning over the past decade has only been possible because of major technological advances in hardware. Without the computational resources and the storage capacity of the systems created in the last decades no system could run today's algorithm. Moreover, much of the progress of recent years was possible due to the improved hardware. Moore's law [29] states that the number of transistors in a dense integrated circuit doubles about every two years and is the only

known physical process following an exponential curve. An analysis by OpenAI shows that since 2012 the amount of compute has even increasing exponentially with a doubling time of 3.4 months [30]. However, the exponential increase seems to come to an end since the size of transistors hit physical limitations. It is assumed that Moore's law will end by around 2025 [31]. Besides the progress in the field and the development of new technology, deep learning models also became better because the number of parameters and the size of datasets grew exponentially. Even the growth in the last five years is astonishing. While the state-of-the-art language model from 2018 [32] had around 94M parameters, the state-of-the-art in 2020 [33] already had 175B parameters. Training such a model on a single V100 GPU would take about 355 years and cost about 4.6M dollars [34]. A recent language model from Microsoft and Nvidia [35] even has 530B parameters. Only a few institutions with massive resources are able to train such big models. In general, inference on low-budget hardware such as smartphones or embedded hardware becomes prohibitive with the growing size of deep networks. Even tough there exist techniques to shrink the model size after training such as quantization [36], model pruning [37], or model distillation [38] it is questionable if making models bigger is the best way to develop intelligent systems.

[31]: Kumar (2015)

[32]: Peters et al. (2018)

[33]: Brown et al. (2020)

[35]: Shoeybi et al. (2020)

Another major issue of deep learning systems is that they suffer from catastrophic forgetting. If a model is trained on a specific task and afterwards trained (or fine-tuned) on another task, the model suffers a "catastrophic" drop in performance over the first task. The reason for this effect is that the model during training on the second task adjusts the parameters learned during the first task and therefore "forgets" the learned mapping functions. Just mixing all datasets or to learn all tasks in parallel in a current multi-task setup [39] doesn't seem feasible to achieve some kind of general intelligence as this involves too many different unrelated tasks. Catastrophic forgetting is also caused by the fact that learning is mostly done offline[8]. Online learning [40] and lifelong learning [41] are currently hot research topics. However, these methods have not yet been established.

8: Offline in this context means that the model parameters are not adapted after training during inference time

Furthermore, there exists problems which may cannot be solved with the current principles used for deep learning. First of all, it is questionable if deep learning models can achieve *real* generalization[9]. With enough data, can achieve generalization in the sense that the model can interpolate within the data distribution. However, deep learning models fail to extrapolate. For example, convolutional neural networks (CNNs) do not generalize to different viewpoints unless they are added to the training data [42].

9: Generalization refers to the ability of the model to adapt properly to previously unseen data from the same distribution

[42]: Madan et al. (2022)

Second, deep learning is not able to learn abstract relationships in a few trials but requires many samples of it and is thus data hungry.[10] Marcus Gary [43] argues that if he tells that a "schmister" is a sister over the age of 10 but under the age of 21, humans can immediately infer whether they or their best friends have any "schmister". However, modern DL systems lacks a mechanism for learning abstractions through explicit, verbal definition and require thousands or even more training samples.

10: Delme!!!

[43]: Marcus (2018)

Third, no DL model has been able to demonstrate causal reasoning in a generic way. Deep learning models find correlations between the

inputs and the outputs, but not the causation. Other AI approaches such as hierarchical Bayesian computing or probabilistic graphical models are better at causal reasoning but cannot be well combined with deep learning models.

Lastly, deep learning models are to some extend too isolated since they have no embodiment and cannot interact with the world. For example, the human body provides needs, goals, emotions, and gut feeling[11]. In current deep learning systems are emotions totally absent and the goals are set externally. Deep Reinforcement Learning can be considered as a first step in the direction of dissolving this isolation, as they interact with a virtual environment. AI systems that interact with the real world do not work well so far. Moravec's paradox [44] states that "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility".

11: one could argue that the body is therefore even a co-processor of the brain

[44]: Moravec (1995)

2.4 Biological Learning

The human brain comprises many interconnected areas processing everything in parallel. For example, Figure Figure 2.3 illustrates the connections between different organizational units in the cerebral cortex which are responsible for vision. It can be seen that these areas are connected in a rather complex structure. Deep learning architectures, on the other hand, are mostly unidirectional and the signal flows unidirectional from layer to layer[12]. However, the choice of the architecture influences the how the model can learn the mapping function from input to output. It could be that the complex structure of our brain comprises an inductive bias which was learned over time through evolution.

12: Except for recurrent connections, skip-connections, or residual-connections

A learning system requires a mechanism that tells the system if something goes well or wrong so that it can learn from it. This is called the *credit assignment problem*. Backpropagation (c.f. Section **??**) solves this problem by propagating the error backwards through the network. However, information flows in the brain only in one direction from the presynaptic neurons to the postsynaptic neurons. Therefore, backpropagation is not biologically plausible. Lillicrap et al. [46] shows that an additional set of random feedback weights is able to transmit useful gradients. Their work has reopend questions how the brain could process error signals and has dispel some long-held assumptions about algorithmic constraints on learning.

[46]: Lillicrap et al. (2016)

Not only the structure of the network and the way how the feedback is calculated is different between biological learning and deep learning. Also the neurons themselves are different. While the artificial neuron doesn't have any dynamics (c.f. Equation (**??**)), biological neurons are highly dynamic: Biological neurons adapt their firing rate to constant inputs, they may continue firing after an input disappears, and can even fire when no input is active (c.f. Section 2.5.4).

Lastly, the neurons in the brain are self-organizing. This means that a group of elementary units such as neurons or a group of neurons perform similar rule of behavior on a sub-set of the available information. Such a system doesn't have a central supervision that orchestrates these

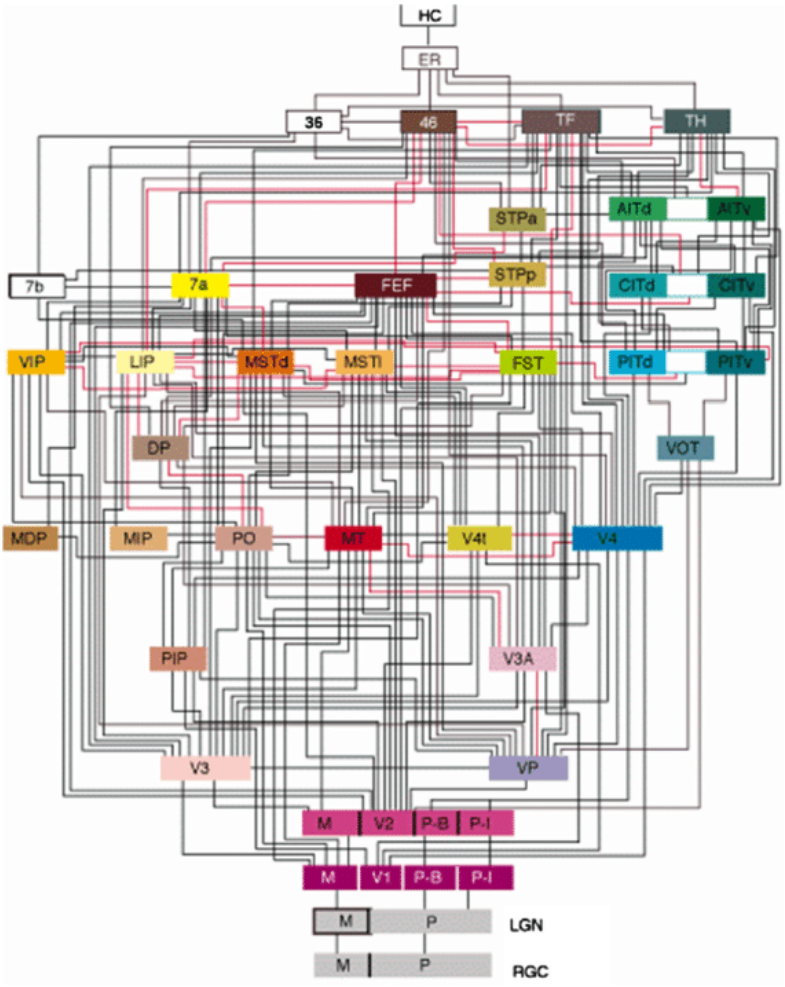

Figure 2.3: The organization of the visual system in the cerebral cortex. The image is from Felleman et al. [45].

units. Each unit applies similar deterministic functions to the information received. Two important principles of such systems are (i) localized learning which means that each unit adapt their behavior to the information they receive; and (ii) emergence which means that there is no explicit loss function that tells the system what to do.

2.5 Neurocomputing

2.5.1 Hebbian Learning

Donald Hebb [47] describes how the connections between cells in the nervous system adapt as: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased". This statement is often simplified to the well-known phrase "Neurons that fire together wire together".

[47]: Hebb (1949)

Hebbian learning is based on this principle. The weight w_{ij} from neuron i to neuron j changes based on the pre-synaptic activity r_i of neuron i and post-synaptic activity r_j of neuron j

$$\Delta w_{ij} = \eta r_i r_j \tag{2.9}$$

where η is the learning rate. Thus, the weights between frequently co-activated neurons becomes strong which is called Hebbian plasticity.

In its original form, Hebbian learning had the problem that the connections could only become stronger but not weaker. Therefore, it is often extended based on the covariance of the activity between neurons. The covariance is positive if two neurons fire often together and negative if they do not often fire together. The following equation changes the weight relative to the covariance:

$$\Delta w_{ij} = \eta(r_i - \psi_i) \cdot (r_j - \psi_j) \tag{2.10}$$

where ψ_i and ψ_j are estimates of the expected pre- and post-synaptic activity[13]. The formulation above lacks of boundaries, i.e. the weights could grow to infinite. A simple solution is to enforce hard boundaries $w_{min} \leq w_{ij} \leq w_{max}$.

13: the expected activity can for example be estimated through a moving average function

Another solution to weaken the connections is given by the Bienenstock-Cooper-Monroe (BCM) learning rule which was introduced by Bienenstock et al. [48] and extended by Intrator and Cooper [49]. They propose a sliding threshold for long-term potentiation (LTP) or long-term depression (LTD) induction. When a pre-synaptic neuron fires and the post-synaptic neuron is in a lower activity state than the sliding threshold, it tends to undergo a LTD (i.e. the connection is weaken).

[48]: Bienenstock et al. (1982)

[49]: Intrator et al. (1992)

Around the same time, Oja [50] improved the learning rule of Equation (**??**) with an normalization term:

[50]: Oja (1982)

$$\Delta w_{ij} = \eta(r_i r_j - \alpha r_j^2 w_{ij}) \tag{2.11}$$

The parameter α is a constant value that determines the size of the norm of the weight vector. This update rule is also known as the Oja learning rule. Furthermore, he has found that a layer of multiple linear neurons converges to the first principle component of the input data. As all neurons only learn the first principle component, a network of multiple neurons in this setting seem not very useful. Differentiation between neurons can be achieved with several different methods. Two well known approaches are the winner-take-all competition (i.e. only the neuron with the most similar activity is selected for learning)[14] and a recurrent circuit that provides a competitive signal (i.e. the neurons compete with their neighbours to become active to learn).

14: in practice is k-winner-take-all often preferred where k instead of one neuron learns

It is known that independent neurons can encode more information and work better than dependent neurons [51]. Anti-Hebbian learning is a method that adds a penalty for similarly active neurons and thus minimizes the linear dependency between neurons. Vogels et al. implemented this by switching the sign of the weight change [52].

[51]: Simoncelli et al. (2001)

[52]: Vogels et al. (2011)

There exists many further improvements for Hebbian learning which are not summarized in this thesis. For example, Joshi and Triesch [53] as well as Teichmann and Hamker [54] adapt the activation function of the

[53]: Joshi et al. (2009)

[54]: Teichmann et al. (2015)

neurons to enforce a certain activity distribution and to stabilize Hebbian learning even in multilayer neural networks.

Similar to large parts of the brain, Hebbian learning is unsupervised and learns based on local information (i.e. neurons in close proximity). However, the brain is also largely recurrent and could guide neighbouring or preceding units. This assumption inspired supervised Hebbian learning. In supervised Hebbian learning, a subset of inputs which should evoke post-synaptic activity can be selected. Supervised Hebbian learning can be extended to top-down and bottom-up learning [55] which leads to a combination of supervised and unsupervised Hebbian learning.

[55]: Grossberg (1988)

2.5.2 Hopfield Networks

Hopfield networks [56] were introduced 1982 by J. Hopfield. They serve as associative (i.e. content-adressable) memory systems. Such systems are particularly useful to retrieve representations based on degraded or partial inputs. Auto-associative memories return for an input the most similar previously seen sample. A classical implementation of an auto-associative memory is the nearest neighbour algorithm [57]. This algorithm compares a given samples with the previously seen training data with a distance metric and returns the most similar sample[15]. Memory networks [58] implement an auto-associative memory within the deep learning framework. Such networks convert an input $\boldsymbol{x}$ to a internal feature representation $I(\boldsymbol{x})$, update memories given the new input $m = G(m, I(\boldsymbol{x}))$, and compute the output features $o = O(m, I(\boldsymbol{x}))$. This process is applied during the training and inference phase. The only difference is that the parameters for the functions I, G, and O are only updated during training.

[56]: Hopfield (1982)

[57]: Fix et al. (1989)

15: or the k most similar samples in the case of the k nearest neighbour (k-NN) algorithm

[58]: Weston et al. (2015)

In a Hopfield network are all neurons are connected, but there are no self-connections: $w_{ii} = 0$ where w_{ij} is the weight between neuron i and neuron j. Furthermore, the weights are symmetrical $w_{ij} = w_{ji}$. A Hopfield network in its original form works only with binary units. For consistency, this networks are called binary Hopfield networks in the following. The output of a neuron in a binary Hopfield network depends on the output of the other neurons within the network:

$$x_i = \sum_{i \neq j} w_{ij} y_j + b \tag{2.12}$$

$$y_i = \begin{cases} 1, & \text{if} x > 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.13}$$

TODO in equation: check where to use y and where to use y with hat

Hopfield networks have their own dynamics and the output evolves over time. If the initial value y_i of a binary Hopfield network has a different sign than $\sum_{i \neq j} w_{ij} y_j + b$ the output will flip (i.e. change its sign). This will in turn influence all other neurons which may also flip. The term $y_i(\sum_{i \neq j} w_{ij} y_j + b)$ is negative if y_i is not equal to $\sum_{i \neq j} w_{ij} y_j + b$, otherwise it is positive. Since the neuron flips if the term $y_i(\sum_{i \neq j} w_{ij} y_j + b)$ is

negative or stays the same if this term is positive, the change of this term can only be positive:

$$\Delta[y_i(\sum_{i \neq j} w_{ij} y_j + b)] \geq 0 \tag{2.14}$$

The negative sum of the term $y_i(\sum_{i \neq j} w_{ij} y_j + b)$ for the entire network is called the energy E of the network:

$$E(\boldsymbol{y}) = -\sum_i y_i \left(\sum_{j>i} w_{ji} y_j + b \right) \tag{2.15}$$

As we have shown in (**??**), the energy function $E\boldsymbol{y}$) of the network can only decrease. Moreover, the energy function has a lower bound and thus the network reaches after a finite number of iterations a stable state. A stable pattern of the network (i.e. no neurons flip their sign) is a local minima of the energy function and is called a point attractor. A pattern is attracted by the closest stable pattern. Thus, the network can be used as associative memory because an input pattern can be matched to the closest stable pattern.

[59]: McEliece et al. (1987)

McElice [59] has shown that a binary Hopfield network with N neurons has a capacity of $C = 0.138N$ (i.e. the number of patterns that can be stored. Hebbian learning (c.f. Section 2.5.1) can be used to store pattern in a Hopfield network. To store a pattern, the weights $\boldsymbol{w}$ must be choosen in a way so that the desired local patterns $(\boldsymbol{y}^1, ..., \boldsymbol{y}^P)$ are local minima of the energy function. By combining Hebbian learning with some smart mathematical transformations it can be shown that the weights can be directly learned with only one iteration over the training patterns[16]:

16: for the derivation of this equation refer to [56]

$$\boldsymbol{w} = \frac{1}{p} \sum_{k=1}^{P} \boldsymbol{y}^k \times (\boldsymbol{y}^k)^T - \boldsymbol{I} \tag{2.16}$$

[60]: Hopfield et al. (1983)

where $\boldsymbol{I}$ is the identity matrix. Later, Hopfield et al. [60] extended the binary Hopfield network so that it can either learn pattern during an awake cycle or forget patterns during a sleep cycle.

[61]: Krotov et al. (2016)

[62]: Demircigil et al. (2017)

One of the limiting factors of binary Hopfield networks is the capacity of $C = 0.138N$. The problems comes from the fact that the energy function is a quadratic function. More than three decades after the introduction of the binary Hopfield networks, Krotov and Hopfield [61] reformulated the energy function as a polynomial function to get polynomial capacity $C \approx N^{a-1}$ where a is the order of the function. Later, the energy function was reformulated as exponential function [62] and thus modern Hopfield networks have an exponential capacity of $C \approx 2^{\frac{N}{2}}$.

[63]: Ramsauer et al. (2021)

The second limiting factor of binary Hopfield networks is that only binary patterns can be stored. Ramsauer et al. [63] extended the binary Hopfield network to continous patterns by reformulating the energy function and the corresponding update rule. Continous Hopfield networks can retrieve continous patterns or even combination of several similar continous patterns. The authors claim that a continous Hopfield networks can replace fully-connected layers, attention layers [64], LSTM layers [65],

support vector machines (SVM) [66], and k nearest neighbor (k-NN) [67].

2.5.3 Spiking Neural Networks

Biological neurons emit spikes. To transmit information, especially the firing rate (i.e. the number of spikes per second in Hz) and precise timing of the spikes are relevant. The amplitude and duration of the spike does not matter much. This behaviour has also been implemented in ANNs. So called Spiking neural networks (SNNs) incorporate the concept of time into their model. SNN do not transmit information in each forward-pass but rather transmit a signal when the membrane potential reaches a threshold value[17]. The neuron fires as soon as the threshold is reached and thereby influences the potential of other neurons. The most prominent model of a spiking neuron is the leaky integrate-and-fire (LIF) neuron [68]. The LIF neuron models the membrane potential with a differential equation. Incoming spikes can either increase or decrease the membrane potential. The membrane potential either decays over time or is reset to a lower value if the threshold value is reached and the neuron has fired. There exists different integrate-and-fire (IF) neurons models such as the Izhikevich quadratic IF [69] or the adaptive exponential IF [70]. While each of these model gas different mathematical properties, the concept of a membrane potential that is increased or decreased through spikes from other neurons and decays over time or by emitting a spike is similar to the LIF.

17: the membrane potential is related to the electrical charge of the membrane of a biological neuron

[68]: Abbott (1999)

[69]: Izhikevich (2003)

[70]: Brette et al. (2005)

Biological neurons have different dynamics. Some neurons fire regularly if the receive an input current, others slow down the firing rate over time or emits bursts of spikes. Modern models of spiking neurons can recreate this behaviour of biological neurons [71].

[71]: Paugam-Moisy (2006)

The synaptic plasticity can be modeled with Hebbian learning (c.f. Section 2.5.1). The spike-timing dependent (STDP) plasticity rule [72] distinguishes the tiring behaviour of pre-synaptic and post-synaptic neurons. If the pre-synaptic neurons fires before the post-synaptic neuron, the connection is strengthen, otherwise it is weaken.

[72]: Bi et al. (2001)

For a long time, SNN only worked for very shallow networks. In 2018, Kheradpisheh et al. [73] has proposed a SNN based on the idea of CNNs called a deep spiking convolutional network. This network uses convolutional and pooling layers with IF neurons instead of classical artificial neurones and is trained with STDP. First, the image is fed into DoG cells. This cells apply the difference of Gaussians (DoG) feature enhancement algorithm. This algorithm subtracts a Gaussian blurred version of an image from the original image. By doing so, positive or negative contrast is detected in the input image. The higher the contrast is, the stronger is a cell activated and the earlier it emits a spike. Thus, the order of the spikes depends on the order of the contrast. These spikes are forwarded to a convolutional layer. Deep spiking convolutional networks use two types of LIF neurons: On-center neurons fire when a bright area is surrounded by a darker area, off-center neurons do the opposite. Convolutional neurons emit a spike as soon as they detect their preferred visual feature[18]. Neurons that fire early perform the STDP update with a winner-takes-all mechanism. This means that the neurons within a layer

[73]: Kheradpisheh et al. (2018)

18: the location of the feature is not relevant as convolution layers are translation invariant

compete with each other and those which fire earlier learn the input pattern. This prevents other neurons from firing and guarantees a sparse connection. Later convolutional layers detect more complex features by integrating input spikes from the previous layer. The features from the last convolutional layer are flatten and a Support Vector Machine is used to classify the features.

2.5.4 Reservoir Computing

As described in Section 2.4, biological neurons are highly dynamical while artificial neurons are not. Reservoir computing introduces such dynamics into an artificial network. Reservoir computing is an umbrella term for networks based on the concepts of Echo State Networks (ESN) [74] and Liquid State Machines (LSM) [75]. A reservoir is a fixed non-linear system that maps a input vector $\boldsymbol{x}$ to a higher dimensional computation space. After the input vector is mapped into computation space, a simple readout mechanism is trained to return the desired output based on the reservoir state. In principle, the system should be capable of any computation if it has a high enough complexity [76]. However, not every system is suited as reservoir. A good reservoir system distributes different inputs into different regions of the computation space [76].

[74]: Jaeger (2001)

[75]: Maass et al. (2002)

[76]: Konkoli (2018)

A ESN is a set of sparsely connected recurrent neurons as visualized in Figure 2.4.

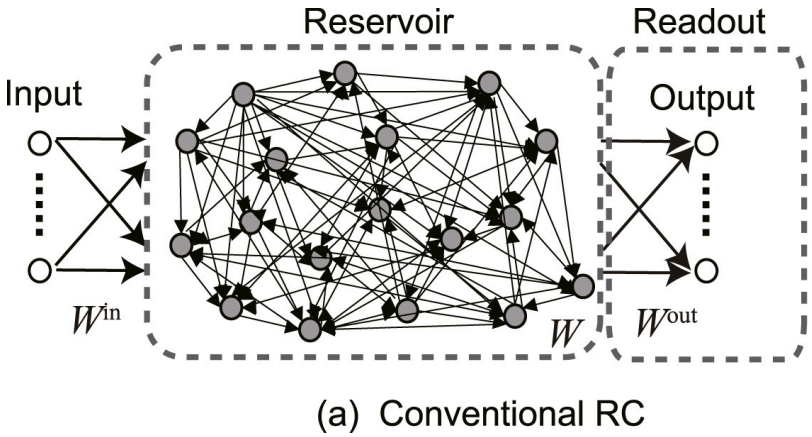

Figure 2.4: Structure of a Echo State Network. The image is from Tanaka et al. [77].

The reservoir consists of N nodes which are connected according a Erdős–Rényi graph model[19] [78]. This graph model is represented by an adjacency matrix $\boldsymbol{w}$[20] of size $N \times N$. The time varying input signal $\boldsymbol{x(t)}$ is mapped to a sub-set of N/M graph nodes by multiplying it with $\boldsymbol{w}_{\text{in}} \in \mathbb{R}^{N \times M}$ and the output by multiplying the reservoir state with $\boldsymbol{w}_{\text{out}} \in \mathbb{R}^{M \times N}$. We refer interested reads to [79] to read more about the mathematical properties and how network is updated in detail.

19: The Erdős–Rényi model is a model for generating random graphs where all graphs on a fixed set of vertices and edges is equally likely

[78]: Erdös et al. (1959)

20: Figure 2.4 uses upper case letter $\boldsymbol{W}$

[79]: Lukoševičius (2012)

In the original form of ESN, only the readout weights are learned, the rest is chosen randomly. The input $\boldsymbol{x(t)}$ brings the recurrent units in a initial state. The recurrent connections inside the reservoir create different dynamics in the network. The readout neurons linearly transform the recurrent dynamics into temporal outputs. The readout weights $\boldsymbol{w}_{\text{out}}$ are trained to reproduce a target function $\boldsymbol{y(t)}$.

Liquid State machines use a spiking neural network instead of a graph of recurrent units as reservoir. The nodes of the spiking neural network are randomly connected together. Thus, every node receives time varying inputs from the inputs as well as from other nodes. The recurrent connections turns the varying input into a spatio-temporal pattern. Similar to ESN, the spatio-temporal patterns of activation are read out by a linear layer.

In general, reservoirs are universal approximators and can approximate any non-linear function given there are enough neurons in the reservoir. They generalize better and faster than equivalent MLP. The main drawback of current systems is that cannot deal well with high-dimensional inputs such as images.

Related Work 3

TODO: Teile auf in related work (zu Beginn) + optionales Kapitel mit weiteren relevanten Themen zu Natural Intelligence, die aber nicht wirklich related work sind sondern eher ein Survey

3.1 Natural Intelligence

This thesis is inspired by the work "A Theory of Natural Intelligence" from von der Malsburg et al. [2]. Therefore, summarize their work in detail in the following.

[2]: Malsburg et al. (2022)

According [2], the process of learning is influenced by "nature", "nurture", and "emergence"[1]. They point out that human genome (as of nature) only contain 1GB of information [80] and humans only absorb a few GB into permanent memory over a lifetime (as of nurture) but it requires about 1PB to describe the connectivity in human brain. Therefore, it is important to distinguish the amount of information to describe a structure from the amount of information needed to generate it. Similar, nature and nurture only require a few GB to construct, respectively instruct the entire human brain. Therefore, they argue that the human brain must be highly structured (i.e. nature and nurture "generate" the human brain by selecting from a set pre-structured patterns). The authors call the process of generating the highly structured network in the human brain the "Kolmogorov [81] Algorithm of the Brain"[2]. Network self-organization is the only mechanism that has not yet been disproved by experiments as the brains Kolmogorov algorithm [82, 83]. This mechanism loops between activity and connectivity, with activity acting back on connectivity through synaptic plasticity until a steady state, called an attractor network, is reached. The consistency property of an attractor network means that a network has many alternative signal pathways between pairs of neurons [84]. Thus, the brain develops as an overlay of attractor networks called net-fragments [85]. Net-fragments consist of small sets of neurons, whereby each neuron can be part of several net fragments. The network self-organization has to start from an already established coarse global structure which is improved in a coarse-to-fine manner to avoid being caught in a local optima.

1: nature refers to the influence of genes and evolution, nurture to the influence of experience and education

[80]: McPherson et al. (2001)

[81]: Kolmogorov (1998)

2: as the Kolmogorov complexity describes the number of bits required by the shortest algorithm that can generate the structure

[82]: Willshaw et al. (1976)

[83]: Willshaw et al. (1979)

[84]: Malsburg et al. (1987)

[85]: Malsburg (2018)

Also, von der Malsburg et al. [2] discuss scene representation (i.e. how a scene is represented in the brain) even tough they point out that this is a contested concept [86]. Scene representation is a organization framework to put abstract interpretation of scene layouts, elements, potential actions, and emotional responses in relation. The details are not rendered as in photographic images but the framework supports the detailed reconstructions of narrow sectors of the scene. The basic goal if learning is to integrate a behavioral schema into the flow of scene representations. They propose the hypothesis that the network structure resulting from self-organization together with the neural activation in

[86]: Freeman III et al. (1990)

the framework of scene representation are the inductive bias that tunes the brain to the natural environment.

Finally, they discuss how net fragments can be used to implement such structures and processes using vision as an example. They point out that a neuron is grouped in one or multiple net fragments through network self-organization. The net fragments can be considered as filters that detect previously seen patterns in the visual input signal. An object is represented by multiple net fragments, where each fragment responds to the surface of that object and has shared neurons and connections with other net fragments representing that object. Thus, net fragments render the topological structure of the surfaces that dominate the environment. Von der Malsburg et al. [2] propose that net fragments represent shape primitives which can adapt to the shape of actual objects[3]. Shifter circuits are one possible implementation of networks that enable invariant responses to the position- and shape-variant representations [87, 88]. They are composed of net-fragments that can be formed by network self-organization [89]. Ref. [2] also argue that net fragments are the compositional data structure used by the brain. A hierarchy of features may be represented by nested net fragments of different size. Complex objects, such as mental constructs, can thus be seen as larger net fragments composed as mergers of pre-existing smaller net fragments.

3: adapt in spite of metric deformations, depth rotation, and position

[87]: Arathorn (2002)

[88]: Olshausen et al. (1995)

[89]: Fernandes et al. (2015)

Von der Malsburg et al. [2] do not address how their interesting theoretical concepts can be implemented in a mathematical model. However, a concrete implementation was done by Claude Lehmann in the form of a Master's thesis [90]. He proposes a new layer called the laterally connected layer (LCL). The LCL layer extends convolutional layers by forming lateral intra-layer connections based on the Hebbian learning rule (c.f. Section Section 2.5.1). Similar to a convolutional filter, the convolutional feature map is calculated. Afterwards, the convolutional feature maps are compared and the lateral impact between feature maps is calculated (i.e. the covariance between the feature maps). When two feature maps are similar in the same pixel locations, their connection strength is increased. By using Hebbian learning, lateral connections are formed between the feature maps with a high lateral impact. Thus, new filters are formed based on existing filters with a high covariance. Lehmann found that LCL layers increase robustness for object recognition. He shows on the MNIST dataset [16] that for a small reduction in accuracy of 1%, the performance on corrupted images increases by up to 21% and that it works especially well for noisy types of corruptions. However, this layers improved performance only on very small networks.

[90]: Lehmann (2022)

In this thesis, the ideas from the work of von der Malsburg et al. [2] are incorporated into a deep learning model. Thus, this work is considered to be *the* inspiration for this thesis. However, this thesis is based on other concepts than the thesis by Lehmann [90] and therefore has little in common with his work.

3.2 Self-Organization

Self-organization is the process by which systems consisting of many units spontaneously acquire their structure or function without interference

from a external agent or system. They organize their global behavior by local interactions amongst themselves. The absence of a central control unit allow self-organizing systems to quickly adjust to new environmental conditions. Additionally, such systems have in-built redundancy with a high degree of robustness as they are made of many simpler individual units. These individual units can even fail without the overall system breaking down.

In nature, groups of millions units that solve complex tasks by using only local interactions can be observed. For example, ants can navigate difficult terrain with a local pheromone-based communication and thus form a collective type of intelligence. Such observations inspired researchers to build algorithms which are based on local communication and self-organization, for example ant colony optimization algorithms [91]. DeepSwarm [92] is a neural architecture search method that uses this algorithm to search for the best neural architecture. This methods achieves competitive performance on rather small datasets such as MNIST [16], Fashion-MNIST [93], and CIFAR-10 [94].

[91]: Dorigo et al. (1997)

[92]: Byla et al. (2020)

Cellular Automata mimic developmental processes in multi-cell organisms. They contain a grid of similar cells with an internal state which is updated periodically. The transition from a given state to a subsequent state is defined by some update rules. During an update, cells are only allowed to communicate with the neighbouring cells. Thus, self-organization is enforced by the definition of the update rules. Such automata can be used to study biological pattern formations [95] or physical systems [96]. Neural Cellular Automata [97] use neural networks to learn the update rule. The input in such a neural network is the state of a given cell and its neighbours, the output the subsequent cell state. Usually, the same network is applied to all cells. In this case, a fully connected neural network which is applied to each cell and its local neighbours can be reformulated as a CNN[98]. NCAs can be trained efficiently with gradient descent to grow given 2D patterns such as images[99, 100]. These images are grown through self-organisation (i.e. the pixels pick a color based on the color of neighboring pixels) and are surprisingly resistant to damage. For example, large parts of the images can be removed and the system is able to rebuild these pixels[4]. However, the aforementioned approaches can only grow the pattern they were trained on. A recent method called Variational Neural Cellular Automata [102] use an NCA as decoder of a Variational Autoencoder [103]. This probabilistic generative model can grow a large variety of images from a given input encoded in a vector format. However, there is still a big gap in performance compared to state-of-the-art generative models. Besides growing 2D patterns, NCAs can also create 3D patterns such as buildings in the popular video game Minecraft by utilizing 3D CNNs [104] or generate structures with specific function such as simulated robots able to locomote[105]. Moreover, self-assembling approaches based on NCAs are not restricted to grid-structures. NCAs can be generalized to graph neural networks [106]. Graph cellular automata (GCA) use graph neural networks [107] instead of CNNs to learn the state transition rules and can thus deal with more complex pattern structures than just 2D and 3D grids. The process of growing images from cells of an NCA can also be inverted. Randazzo et al. [108] propose to use NCA to classify given structures such as images. They apply the same network to each pixel

[95]: Wolfram (1984)

[96]: Vichniac (1984)

[97]: Wulff et al. (1992)

[98]: Gilpin (2019)

[99]: Mordvintsev et al. (2020)

[100]: Mordvintsev et al. (2022)

4: a demo of this regeneration process is available at [101]

[102]: Palm et al. (2022)

[103]: Kingma et al. (2014)

[104]: Sudhakaran et al. (2021)

[105]: Horibe et al. (2021)

[106]: Grattarola et al. (2021)

[107]: Zhou et al. (2021)

[108]: Randazzo et al. (2020)

and its neighbours of an image. In an iterative process based on local communication, the image fragments then agree on which object they represent. NCAs can even be used to control reinforcement learning (RL) agents. Variengien et al. [109] use the observations of the environment as state of the NCA, the subsequent state predicted by the NCA are used as Q-value estimates of a deep Q-learning algorithm [110].

[109]: Variengien et al. (2021)

[110]: Mnih et al. (2013)

Self-organization can not only be used to generate structures but also to optimize the weights of a neural networks over the agents lifetime. For example, a Hebbian learning rule for meta-learning can be used to self-organize the weights of a RL agent over his lifetime[111]. This means that across multiple episodes the weights of a Hebbian based model are learned. The weights of the agents policy are reset in every episode and the Hebbian based model is used to update them. This allows the agent to adapt better to the changed conditions within the environment.

[111]: Najarro et al. (2020)

Besides optimizing the weights, self-organization has also been used to change the learning rule itself. The method "Evolve and Merge" [112] uses the so called "ABCD" Hebbian learning rule which updates the weights as follows:

[112]: Pedersen et al. (2021)

$$\Delta w_{ij} = \alpha(Ao_io_j + Bo_i + Co_j + D) \tag{3.1}$$

α is the learning rate, o_i and o_j are the activity levels of connected neurons and A, B, C, and D are learned constants. For each connection in the network is one learning rule initialized and the constants are learned. After a pre-defined number of epochs, the learning rules are clustered and the ones with similar constants are merged. By repeating this process, the number of parameters can be reduced and robustness increases according to the authors.

Alternatively, it is also possible to initialize the network with shared parameters instead of starting with many rules and merging them over time. Kirsch and Schmidhuber [113] use multiple tiny recurrent neural networks (RNNs) that have the same weight parameters but different internal states[5]. By using self-organization and Hebbian learning, they show that it is possible to learn powerful learning algorithms such as backpropagation while running the network in forward-mode mode only. However, it works only for small-scale problems as it can get stuck in local optima. In general seem self-organizing systems to be hard to optimize and only to work for small datasets or simple problems so far.

[113]: Kirsch et al. (2021)

5: Intuitively, these tiny RNNs can be interpreted as more complex neurons.

Risi [114] describes why self-organizing systems are hard to train; First, the system is hard to control because there is no central entity in charge but the system must still be nudges into the right direction. Second, self-organizing systems are unpredictable (i.e. there exist no mathematical model that tells the outcome of the self-organizing process).

[114]: Risi (2021)

3.2.1 Growing Networks

Unsupervised learning techniques usually map high dimensional input data to a lower-dimensional representation. One approach to do so are self-organizing maps (SOM) [115, 116]. They map the input data to a discretized representation of the input space of the training samples,

[115]: Kohonen (1982)

[116]: Kohonen (1989)

called a map. In opposite to ANNs, they use competitive learning instead of error correction learning (i.e. back-propagation with gradient descent). A weight vector is used to map the data to a node in the mapping field. The datapoints "compete" for the weight vectors. The weight vector of a node in the map that best matches a datapoint is moved closer to that input, as are nodes that are in the neighbourhood. By doing so, samples that are close in the input space are also closed in the resulting maps.

However, SOM have two major limitations; First, the network structure must be pre-defined which constraints the result mapping accuracy. Second, the capacity of the map is predefined through the number of nodes. Growing networks are able to overcome this limitations. Growing networks add nodes or whole layers of nodes into the network structure at the positions of the map where the error is highest. Many growing networks [117–119] add such units after a fixed number of iterations in which the error is accumulated. After adding a unit, it takes several iterations to accumulate the error again until the next node can be added.

[117]: Fritzke (1994)

[118]: Reilly et al. (1982)

[119]: Fritzke (1994)

Grow When Required (GWR) networks [120] use a different criterion to add nodes. Instead of adding nodes to support the node with the highest error, nodes are added when a given input samples cannot be matched with the current nodes by some pre-defined accuracy. This allows the network to adapt the growing process rather fast; The networks stops growing when the input space is matched b the network with some accuracy and the networks starts growing again if the input distribution changes.

[120]: Marsland et al. (2002)

Such GWR networks can be used to build self-organizing architectures. For example, Mici et al. [121] build a self-organizing architecture based on GWR to learn human-object interactions from videos. They use two GWR in parallel, one to process feature representations of body postures and another to process manipulated objects. A third GWR is used to combine these two streams and to create action–object mappings in a self-organized manner. By doing so, they are able to learn human-object interactions and exhibit a model which is more robust to unseen samples than comparable deep learning approaches.

[121]: Mici et al. (2018)

3.2.2 Self-Organization in Spiking Neural Networks

Spiking neural networks (SNNs) (c.f. Section Section 2.5.3) communicate through binary signals known as spikes and are very efficient on special event-based hardware[122]. There exist several methods to self-organize such architectures. For the sake of completeness, two well-known approaches are described in the following. However, since this thesis focuses on self-organization in deep learning systems, these approaches are only roughly described and for detailed explanations please refer to the respective literature.

[122]: Davies et al. (2018)

Similar to deep learning, there exists a multitude of different network architectures; Shallow [123, 124] and deep networks [125, 126] structures, fully connected [127] and convolutional layers [128, 129], as well as based on different learning rules such as supervised [130, 131], unsupervised [127, 132] and reinforcement learning based [133].

[123]: Masquelier et al. (2007)

[124]: Yu et al. (2013)

[125]: Kheradpisheh et al. (2018)

[126]: Mozafari et al. (2019)

[127]: Diehl et al. (2015)

[128]: Cao et al. (2015)

A representable method for self-organization in SNNs is proposed by Raghavan et al. [134]. They introduce a stackable tool-kit to assemble multi-layer neural networks. This tool-kit is a dynamical system that encapsulates the dynamics of spiking neurons, their interactions as well as the plasticity rules that control the flow of information between layers. Based on the input, spatio-temporal waves are generated that travel across multiple layers. A dynamic learning rule tunes the connectivity between layers based on the properties of the waves tiling the layers[6].

[134]: Raghavan et al. (2020)

6: for more information please refer to [134]

An alternative method proposed by Raghavan and Thomson [135] grows a neural network. They start with a single computational "cell" and use a wiring algorithm to generate a pooling architecture in a self-organizing fashion. The pooling architecture emerges through two processes; First, a layered neural network is grown. Second, self-organization of its inter-layer connections is used to form defined "pools" or receptive fields. They us the Izikhevich neuron model [69] in the first layer to generate spatio-temporal waves. The units in the second layer learn the "underlying" pattern of activity generated in the first layer. Based on the learned patterns, the inter-layer connections are modified to generate a pooling architecture[7].

[135]: Raghavan et al. (2019)

[69]: Izhikevich (2003)

7: for more information please refer to [135]

In general, SNNs have to "convert" static input data such as images to a dynamic signal. For example, images are often converted to such signals by using Difference of Gaussian (DoG) convolution filters [73, 136]. Such filters subtract one Gaussian blurred version of an original image from another, less blurred version[8]. This subtraction results in spikes for each pixel. To encode the filter output into a temporal signal, bigger spikes are forwarded earlier in time than smaller spikes. However, such approaches lose a lot of information about the input. For example, in the process described above are all information about color and thin structures lost. To the author of this thesis, this seems to be the reason why these SNNs can't match the performance of deep learning algorithms so far and often only work well for small gray-scale image-datasets such as MNIST [16].

[136]: Vaila et al. (2019)

[73]: Kheradpisheh et al. (2018)

8: DoG filter can thus be used to reduce noise and to detect edges

3.2.3 Relevance

Self-organization is, according to von der Malsburg et al. [2], one of the key elements for natural intelligence. Therefore, this concept of local optimization and interaction plays a very important role in this thesis. Specifically, two types of self-organization are proposed in this thesis; Vertical self-organisation is based on layer-wise learning. In this approach, the layers of a neural network are considered separate units and are updated independently. The communication between these units takes place by feeding data sequentially through the layers. The second type of self-organization is vertical self-organisation. In this approach, data is divided into patches and then analysed by independent networks. Thus, small networks form the self-organising units and communicate with each other by adding the output of one network to the input of another network. These two types of self-organization are described in more detail in Chapter 4.

[2]: Malsburg et al. (2022)

3.3 Alternative Training Algorithms

Neural networks, especially deep neural networks, are usually optimised by backpropagation of errors (c.f. Section Section 2.2). Soon after backpropagation was published, it was question whether this algorithm is suitable to explain the learning in the brain [137, 138]. Besides the fact that backpropagation seems no biologically plausible and is responsible for many of the limitations described in Section Section 2.3, it also has technical shortcomings. First, gradients might vanish or explode when propagated through too many layers [139]. Second, it turns the networks into "black boxes" and prevents the users from getting useful insights from trained models[9]. Training end-to-end does not allow to control the effect of the loss function on a hidden layer because of the non-linearity in the network [140]. Finally, the loss landscapes are non-optimal which might lead to slow convergence and the possibility that the reached local minimum is suboptimal [15].

[137]: Crick (1989)

[138]: Grossberg (1987)

[139]: Zhang et al. (2020)

9: training a model a layer-wise allows modularization, i.e. divide an conquer

[140]: Lee et al. (2015)

[15]: Ioffe et al. (2015)

Hebbian learning (c.f. Section Section 2.5.1), on the other hand, is widely accepted in the fields of neurocomputing, psychology, neurology, and neurobilogy [141]. Hebbian mechanisms are for example sufficient for topographic mappings[10] [116, 142], neuroplasticity[11] [143], or recognizing non-linear patterns [144], but also have limitations [145]. Hebb's learning rule is insufficient as general rule and is limited temporally as it requires (almost) synchronous stimuli [146][145]. However, many signals such as motor control problems, are sequential and Hebbian learning has therefore to be combined with additional memory mechanisms [147].

[141]: Widrow et al. (2019)

10: mapping input data to a discretized representation

[116]: Kohonen (1989)

[142]: Grajski et al. (1990)

11: the networks ability to reorganize itself by forming new connections

[143]: Montague et al. (1991)

[144]: Mel (1992)

[145]: Anderson (1998)

[146]: Rumelhart et al. (1987)

[147]: Grossberg et al. (1989)

There exists many alternative learning functions to these two well known algorithms that might overcome the aforementioned limitations. A group of algorithms that do not require end-to-end forward or backward pass is called *Proxy Objective*. These algorithms use a proxy objective function for each layer. While backpropagation calculates a global loss after the forward pass, this strategy uses a proxy function for each layer. The loss function of the first Layer L_1 (i.e. the proxy objective of the first layer) influence only the trainable parameters of the first layer θ_1 but not the parameters of the second layer θ_2. This allows to decouple the layers. The different instances of *Proxy Objective* methods mainly differ in the proxy objective function. However, the idea behind the proxy objective function is always the same; namely to characterize the separability of the hidden representations. Minimizing L_1 encourages the first layer to improve the separability of its output representations, making the classification problem simpler for the subsequent layer. The separability between hidden representation vectors can be measured by distance/similarity metrics as done in [148, 149]. Wang et al. [150] propose an auxiliary neural network to map the hidden representation to another feature space before computing their distance/similarity. This allows to choose the dimensionality of the feature space in which the proxy objective function is computed regardless of the given network layer dimension. Others [151–153] propose an additional network with either fixed or trainable weights as proxy function and used this auxiliary network's accuracy to quantify data separability. Nokland and Eidnes [154] combine the two aforementioned approaches by using two auxiliary networks; one network is trained with a "similarity matching loss" and the other with a cross-entropy loss (i.e. to quantify separability with an auxiliary classifier

[148]: Duan et al. (2022)

[149]: Duan et al. (2020)

[150]: Wang et al. (2021)

[151]: Belilovsky et al. (2019)

[152]: Mostafa et al. (2018)

[153]: Marquez et al. (2018)

[154]: Nokland et al. (2019)

accuracy). A major issue of the first *Proxy Objective* methods is that the models are trained layer-wise[12]. The resulting computational complexity can be reduced with synchronous or asynchronous training. In the synchronous setting is first a forward pass through all layers performed and afterwards are all layers trained simultaneously by minimizing their own proxy objective function [151]. In the asynchronous settings are all layers trained simultaneously by using a replay buffer for each layer that stores the layers output [155]. Each layer receives its input from such a buffer instead of the previous layer what eliminates the need for an end-to-end forward pass.

12: when layer k is trained, the layers $1, \ldots, k-1$ are frozen and the layers $k+1, \ldots, n$ are not updated

[155]: Belilovsky et al. (2020)

[156]: Hinton (2022)

Recently, Hinton [156] introduced the *Forward-Forward (FF) Algorithm* that uses a proxy objective functions over multiple time-steps. This algorithm uses two forward passes, one with positive (i.e. real) data and one with negative data. Each layer has its own objective function which is to have a high "goodness" for positive data and a low "goodness" for negative data. The goodness is measured by the sum of the squared neural activities. The goal of the learning process is to make the goodness be above some threshold value for real data and below that threshold for negative data. More specifically, the aim is to correctly classify input vectors as positive data or negative data. The probability that an input vector is positive $p(positive)$ is given by applying the logistic function σ to the goodness minus the threshold θ

$$p(positive) = \sigma\left(\sum_j y_j^2 - \theta\right) \tag{3.2}$$

where y_j is the output of a hidden unit. To perform a classification task, the label is included in the input. Positive data consists of an input vector (e.g. an image) and the correct label, negative data consists of an input vector with the wrong label. The goal is then to have a high goodness in each layer for the positive data and a low goodness for the negative data. The only difference between positive and negative data is the label and thus the *FF* algorithm only learns features that correlate with the label. During inference, the data is fed with each label through the network and the one with the highest goodness is kept as prediction. One major weakness is that one layer at a time is learned and later layers cannot affect what is learned in earlier layers. This limitation can be overcome by treating the static input vector as a sequence (i.e. use the same input for multiple steps) and by using a multi-layer recurrent network [157]. The algorithm still runs forward in time but the activity vector at each layer is determined by the activity vector of the previous layer and the activity vector of the subsequent layer at the previous time-step.

[157]: Hinton (2021)

Target Propagation methods are a group of algorithms that require an end-to-end forward pass but not an end-to-end backward pass. These algorithms compute targets[13] rather than gradients at each layer. Similar to gradients, the targets are propagated backward through the network. While the output layer obviously uses the true label as its target (i.e. the goal is to predict the label), the targets of previous layers are found sequentially (from the output layer to the input layer). Good targets are those that minimize the loss in the output layer if they are realized in the forward pass. The simplest solution would be to apply the inverse function of each layer to propagate the target backward. If h_l are the

13: a target can be interpreted as the value that should be predicted by a layer

activations and θ_l the parameters of layer l, we can define the forward pass as:

$$h_l = f(h_{l-1}; \theta_l) \tag{3.3}$$

The inverse function that yields the target activation $\hat{h}_l$ would then be:

$$\hat{h}_l = f^{-1}(\hat{h}_{l+1}; \theta_{l+1}) \tag{3.4}$$

As mentioned before, the target of the final layer $\hat{h}_L$ is the loss-optimal output activation such as the correct label distribution. However, advanced neural networks are usually not invertible and approximate inverse transformations have to be learned with decoders

$$g(h_{l+1}; \lambda_{l+1}) \approx f^{-1}(h_{l+1}; \theta_{l+1}) \tag{3.5}$$

where λ_l are the trainable parameters of the decoder at layer l. Thus, the target activation can be approximated by an encoder:

$$\hat{h}_l \approx g(\hat{h}_{l+1}; \lambda_{l+1}) \tag{3.6}$$

Different instances of *Target Propagation* methods mainly differ in the way the the targets are generated and/or the loss function of the decoder L_g. Vanilla Target Propagation [158] directly derives the target from the decoder $\hat{h}_l = g(\hat{h}_{l+1}; \lambda_{l+1})$ and trains the decoder $g()$ by minimizing

[158]: Bengio (2014)

$$L_g = \|g(h_{l+1} + \epsilon; \lambda_{l+1}) - (h_l + \epsilon)\|_2^2 \tag{3.7}$$

where ϵ is some added Gaussian noise to enhance generalization. It is obvious that this loss functions encourages the decoder $g()$ to learn the inverse of $f()$ by minimizing the difference between the activations from the forward-pass h_l of layer l and the decoder's output (based on the activations of the subsequent layer h_{l+1}). The targets are predicted by feeding the *target activations* $\hat{h}_{l+1}$ through the decoder[14].

14: note that these are two different kind of activations: The inverse function is learned based on the activation of the forward pass h_l, the target activation is predicted based on the loss-optimal output activation $\hat{h}_L$

Later, a major improvement was to extend the decoder $g()$ of Vanilla *Target Propagation* with a correction term that enables a more robust optimality guarantee for bigger networks [159]. This method is known as *Difference Target Propagation* and extends Equation (**??**) as follows:

[159]: Lee et al. (2015)

$$\hat{h}_l = g(\hat{h}_{l+1}; \lambda_{l+1}) + [h_l - g(h_{l+1}; \lambda_{l+1})] \tag{3.8}$$

The extra term $[h_l - g(h_{l+1}; \lambda_{l+1})]$ is to correct errors of the decoder in estimating the inverse. *Difference Target Propagation* has been further improved by novel loss functions L_g for the decoder. These methods are known as *Difference Target Propagation with Difference Reconstruction Loss* and *Direct Difference Target Propagation* [160]. However, since the concepts are roughly the same as for *Difference Target Propagation* (despite adding some terms to the loss function), I do not summarize these methods in more detail here. An advantage of calculating targets is that only layer-wise gradients are required what allows models to have

[160]: Meulemans et al. (2020)

non-differentiable operations [159]. A major drawback is that auxiliary models have to be trained to calculate the hidden targets $\hat{h}_l$ what might overweight the savings achieved by eliminating a full backward pass.

Another group of alternative learning algorithms are known as *Synthetic Gradients* [161, 162]. *Synthetic Gradients* methods replace the gradients used in the backward pass by approximating local gradients with auxiliary models and utilize gradient-based optimization algorithms such as SGD to update the weights locally. The auxiliary models are usually fully-connected networks that are trained to regress a layer's gradients when given the layer's activations. End-to-end backwards passes are only performed occasionally to acquire real gradients that can be used to trained the gradient approximation models. Thus, the frequency of end-to-end backward passes is reduced. When auxiliary input models are used to predicts the layer's input, the frequency of the forward pass can be reduced as well in the same manner as for the backward pass [161]. It is also possible to get rid of the backward pass completely by training the auxiliary gradient models with local information only [163]. However, this works significantly worse than when real gradients are used. An advantage of *Synthetic Gradients* methods is that they can be used to approximate backpropagation through time (BTT) for an unlimited number of steps [161]. It has been shown that this allows more efficient training for learning long-range dependencies compared to BTT.

[161]: Jaderberg et al. (2017)

[162]: Czarnecki et al. (2017)

[163]: Lansdell et al. (2020)

Many other methods are motivated purely by biological plausibility. Some examples are [46, 164–167]. However, these biological plausible methods have been significantly outperformed by backpropagation of error on meaningful benchmark datasets [168]. In the following *Feedback Alignment* methods are discussed as they seem to be the most popular group of biological plausible alternatives to backpropagation. From a biological point of view, one of the major criticisms of backpropagation is the "weight transportation problem" [46]; it is believed that the human brain doesn't have a backward pass where an error signal is passed to previous layers. *Feedback Alignment* algorithms use fixed, random weights during the backward pass [46]. Thus the symmetry between the weights used during the forward pass and the backward pass is broken. Later, the algorithm was improved by using fixed, random weights that share the signs with the actual weights of the network [167].

[168]: Bartunov et al. (2018)

[46]: Lillicrap et al. (2016)

[167]: Liao et al. (2016)

Lastly, *Auxiliary Variables* are another group of learning algorithms [169–172]. These methods use the idea of variable splitting, i.e. transform a complicated problem into a simpler one by introducing additional trainable variables. Introducing auxiliary variables may pose scalability issues, and these methods require special, usually tailor-made solvers. Therefore, such methods are not reviewed in more detail in this thesis.

[173]: Duan et al. (2022)

[168]: Bartunov et al. (2018)

According to [173] and [168], only proxy objective functions have achieved competitive performance with large models on large datasets such as ImageNet [174]. The other approaches work only on smaller datasets such as MNIST [16] or CIFAR-10 [94] so far.

End-to-end backpropagation is not compatible with local self-organisation which is one of the key concepts of natural intelligence according to von der Malsburg et al. [2]. With end-to-end backpropagation, systems are trained as a single unit and consequently have no independent local units that can organise themselves. Therefore, alternative learning algorithms

[2]: Malsburg et al. (2022)

is very relevant to this thesis. In fact, the approach based on vertical self-organisation uses proxy objective functions to train the layer locally as independent units (c.f. Chapter 4). Proxy objective functions are used as these are the only alternative learning algorithm known that scale well on large neural networks and large datasets.

3.4 Visual Representation Learning

One of the earliest methods of learning visual representations are *Autoencoders*. *Autoencoders* have been introduced 1985 [28] and learn an encoder and a decoder function [175]. The encoder A maps the input from a high-dimensional space to a lower dimensional embedding space $A : \mathbb{R}^n \rightarrow \mathbb{R}^e$ and the decoder B reverses this mapping $B : \mathbb{R}^e \rightarrow \mathbb{R}^n$. Typically, neural networks are used to learn the encoder function A and decoder function B by minimizing a reconstruction loss [176]. In order that the functions A and B are not just the identity operators, some regularization is needed. One of the simplest regularization methods is to introduce a bottleneck, i.e. to compress the representation with the encoder while still being able to re-create the original input as good as possible with the decoder. With a bottleneck layer, the number of neurons is limited. An alternative (or a supplement) is to limit the number of activations by enforcing sparsity. Autoencoders with such a sparsity constraint are also known as *Sparse Autoencoders*. A sparsity constraint can be imposed with L_1 regularization or a kullback-leibler (KL) divergence between expected average neuron activation and an ideal distribution [177]. Other popular versions of *Autoencoders* are *Denoising Autoencoders* [178], *Contractive Autoencoders* [179], and *Variational Autoencoders (VAE)* [103]. In *Denoising Autoencoders*, the input is disrupted by noise (e.g. by adding Gaussian noise or removing some pixels by using Dropout) and fed into the encoder A. The goal of the decoding function B is to reconstruct the clean version of the input without the added noise. By doing so, the *Denoising Autoencoders* learns to create more robust representation of the input or can be used for error correction. *Contractive Autoencoders*, on the other hand, try to make the feature extraction less sensitive to small perturbations. By adding the squared Jacobian norm to the reconstruction loss, the latent representations of the input tend to be more similar to each other. This diminishes latent representations that are not important for the reconstruction and only important variations between the inputs are kept. The encoder of *Variational Autoencoders* map the input to a probabilistic distribution; Instead of mapping the input to an encoding vector, the input is mapped to a vector representing the means μ and another vector representing the standard deviations σ. This done by adding two fully connected layers after the encoder A, one layer to predict μ and the other layer to predict σ. Afterwards, a variable is sampled from this distribution $z \sim \mathcal{N}(\mu, \sigma^2)$[15] and fed into the decoder B. By doing so, the latent space becomes by design continuous and allows random sampling and interpolation of data.

[28]: Rumelhart et al. (1985)

[175]: Baldi (2012)

[176]: Ranzato et al. (2007)

[177]: Le et al. (2012)

[178]: Vincent et al. (2008)

[179]: Rifai et al. (2011)

[103]: Kingma et al. (2014)

15: this process is also called the re-parameterization trick

Other approaches for *self-supervised learning* typically augment the visual scene and either predict the augmentation parameters, reconstruct the original version of the image, or learn consistent representations among augmented views of the image. For example, some models use masking

to learn visual representation; A part of the input image is removed and the model predicts the missing part (image inpainting) [180]. This not only allows to generate part of images but also to learn visual representations of the image [181–183]. Other approaches rotate images and predict the rotation angle to generate representations [184, 185]. There exist many other methods that augment images and learn good visual representations based on it. A comprehensive overview is given by [186].

[181]: Pathak et al. (2016)

[182]: He et al. (2022)

[183]: Shi et al. (2022)

[184]: Komodakis et al. (2018)

[185]: Zhai et al. (2019)

[186]: Chen et al. (2022)

Another popular *self-supervised learning* paradigm is *contrastive learning*. The idea of *contrastive learning* is that similar images should yield similar representations. Typically, *contrastive learning* models are trained without labels. In this case, two augmented views of the same image are created, fed through an encoder and the representation of this two views are pushed together in the embedding space by maximizing their similarity [187–189]. Subsequently, the encoder can be used to generate image representations that are used for other downstream tasks such as image classification. However, *contrastive learning* can also leverage information from annotations. For example, Khosla et al. [190] use labels in a contrastive setting to pull the representations of images of the same class together in the embedding space, while simultaneously pushing apart clusters of samples from different classes.

[187]: Chen et al. (2020)

[188]: He et al. (2020)

[189]: Caron et al. (2020)

[190]: Khosla et al. (2020)

In this thesis, representations of images are learned. Two different principles are applied; vertical self-organisation uses a type of contrastive learning by forcing samples from the same class to have similar representations and samples from different classes to have high diversity. Horizontal self-organisation uses independently trained variational autoencoders to obtain good representations of input patches.

3.5 Meta-Learning

TODO: NOT SURE IF THIS CHAPTER IS STILL NEEDED -> ADD REFERENCE TO THIS CHAPTER

Meta-learning is the process of distilling experience from multiple learning cycles and using the accumulated experience to improve the future learning process [191]. Therefore, meta-learning is also referred to as "learning-to-learn" [192] and improves learning on a lifetime (i.e. single agent) and evolutionary timescale (i.e. population of agents) [193]. Typically, meta-learning includes *base learning* (also referred to as *inner* or *lower* learning) and *meta-learning* (also referred to as *outer* or *upper* learning). During *base learning*, the model learns a typical task such as image classification based on a dataset and a objective function. During *meta-learning*, an algorithm updates the *base learning* algorithm based on a meta objective function[16]. Thus, meta-learning usually iterates between learning a task and improving the learning algorithm that is used to learn the task.

[191]: Hospedales et al. (2021)

[192]: Thrun et al. (1998)

[193]: Schrier (1984)

16: Typical meta objective function aim to improve generalization or learning speed

In the following, I use the categorisation according to Hospedales et al. [191] to describe the different aspects of meta-learning algorithm.

Meta-Representation What meta-knowledge shall be learned by the outer *meta-learning* algorithm.

Meta-Optimizer How the outer *meta-learning* algorithm learns, i.e. the from of the learning algorithm.

Meta-Objective What the goal of the *meta-learning* algorithm is.

The first dimension of the meta-learning landscape is the **meta-representation**. Meta-representations describe which part of the learning strategy should be learned. One possibility is to learn good *initial parameters* that can be used by the model during *base learning*. Good initial parameters are only a few gradient steps away from a set of parameters that can solve a task $\mathcal{T}$ drawn from a set of tasks $p(\mathcal{T})$. A popular algorithm to learn initial parameters is called MAML [194, 195] and works well on smaller networks. However, a major challenge is that the outer *meta-learning* algorithm has to find a solution for as many parameters as the inner *base learning* needs. Therefore, many approaches focus on isolating a subset of parameters to meta-learn [196–198]. *Black-box models*, on the other hand, use *meta-learning* to directly provide the parameters required to classify data (i.e. the *base learning* algorithm maps a sample directly to class without iterative parameter optimization) [199–201]. A special case of the black-box models are the approaches based on *metric learning*[17]. The outer *meta-learning* process optimizes a model to transform inputs into representations that can be used for recognition by similarity comparison (e.g. by using cosine similarity or euclidean distance) [197, 202, 203].

[194]: Finn et al. (2017)

[195]: Finn et al. (2018)

17: an explanation why this approach can be considered black-box learning is provided by Hospedales et al. [191]

Other approaches learn the *optimizer* of the inner *base learning* algorithm [204–206]. Typically, such approaches generate each optimization step of the *base learning* algorithm based on the models parameter and a given base objective function. The trainable component can for example be simple hyper-parameters such as the learning rate [206] or more sophisticated such as pre-conditioned matrices [207]. Even the learning algorithm itself can be learned [113].

[113]: Kirsch et al. (2021)

The outer *meta-learning* algorithm can also be used to learn the *inner objective function* (while the outer objective function is fixed). Such loss-learning approaches output a scalar value that is treated as loss by the inner optimizer based on relevant quantities such as prediction/ground truth pairs or model parameters. Common goals of this approach are to obtain a loss that has less local minima [208, 209], provides better generalization [210–212], leads to more robust models [213], or to learn from unlabelled data [214, 215].

Other approaches use the outer *meta-learning* loop to learn *network architectures* for the inner *base learning* cycle. Some approaches use reinforcement learning in the outer loop to learn CNN architectures [216], while other approaches use evolutionary algorithms to learn the topology of LSTM cells [217] or to model the network by a graph of network-blocks [218].

The outer *meta-learning* loop can also meta-learn hyper-parameters such as regularization-strength [219, 220], task-relatedness for multi-task learning [221], or sparsity-strength [221]. Furthermore, it can meta learn suitable data augmentation methods [222, 223], or improve the selection process for the samples of a mini-batch [224, 225]. Even the dataset itself can be learned with dataset distillation; A rather large datasets can be summarized a smaller dataset with only a few support images [226, 227] that still allow good generalization on real test images. In sim2real

learning [228], also the graphics engine [229, 230] can be trained in an outer loop so that the performance on real-world data is maximized.

The second dimension of the meta-learning landscape is the **meta-optimizer**. The meta-optimizer describes how the algorithm in the outer *meta-learning* loop is learned. Many methods use backpropagation of error and gradient descent (c.f. Section Section 2.2) to optimize the meta parameters [194, 204, 213, 219–221, 227]. However, this algorithm has some well-known downsides (c.f. Section XXXXXXXXXX) and requires differentiability. When the inner *base learning* algorithm includes non-differentiable steps [222] or the outer meta objective function is non-differentiable [231], many methods utilize the RL paradigm to optimize the outer objective [232]. However, using RL to alleviate requirement for differentiability is usually computationally very costly. Finally, evolutionary algorithms can be used for learning the outer *meta-learning* function [233–235]. These algorithms have no differentiability constraints, do not suffer from gradient degradation issues, are highly parallelizable, and can often avoid local minima better than gradient-based methods [235]. The downside of evolutionary algorithms is that often a large population size is required (especially if many parameters have to be learned) and the performance is generally inferior to gradient-based methods for large models. These three learning methods are also used in conventional machine learning. However, meta-learning comparatively uses RL and evolutionary algorithms more frequently as some components are often non-differentiable in the meta-learning setup.

The third dimension of the meta-learning landscape is the **meta-objective**. The meta-objective describes what the goal of the meta-learning algorithm is. Typically, the performance of a meta-learning algorithm is evaluated with a metric on the inner loop or a meta-metric on the outer loop. However, there are several design options within the meta-learning framework. First, the inner base-learning episodes can either take few [194, 204] or many [221, 236] samples and thus the goal can be to either improve few- or -many-shot performance. Second, the validation loss of the inner base-learning algorithm can either be calculated at the end of a learning episode to encourage a better *final* performance of the base task or as sum of the loss calculated after each update step to encourage *faster* learning [237]. Third, the goal of the meta-learning can either be to better solve any task drawn from a set of (often related) tasks (i.e. multi-task setting) [194, 202, 213], or to solve one specific task better than when only the inner base learning algorithm is used (i.e. single task setting) [221]. Finally, the meta-optimization can either be done offline (i.e. the inner and outer loop alternate) [194] or online (i.e. the meta-learning takes place within a base learning episode) [213].

4 Methods

This chapter describes the methodology used. A large part of the contribution of this thesis consists of identifying appropriate findings from neuroscience and adapting them to a deep learning setting. This identified concept and the link between the two disciplines is described in chapter 4.1. It should be noted that this section presents only one possible interpretation for the implementation of neuroscientific concepts in the context of deep learning and that alternative interpretations might also be promising. Afterwards, two possible implementation approaches are described, which differ mainly in the type of local self-organisation. These types of self-organisation are referred to as horizontal and vertical self-organisation and are described in Section 4.2 and Section 4.3 respectively.

4.1 Neuroscientific Concepts

4.1.1 Self-Organisation

It is known that large parts of the human brain are self-organizing [238]. Self-organization is the process by which systems consisting of many units acquire their function through local interaction and without interference from a external supervisory system. Recently, renowned scientists [2] put forward the hypothesis that this process of self-organization is *the* key mechanism of natural intelligent systems such as the human brain. Dresp [239] describes seven clearly identified properties of self-organization in the human brain: (i) modular connectivity, (ii) unsupervised learning, (iii) adaptive ability, (iv) functional resiliency, (v) functional plasticity, (vi) from-local-to-global functional organization, and (vii) dynamic system growth. However, it is not obvious how these insights from neuroscience can be integrated into a deep learning framework.

[238]: Kelso (1995)

[2]: Malsburg et al. (2022)

[239]: Dresp (2020)

Deep learning networks are usually optimized with end-to-end backpropagation of error. Thus, the entire network is optimized for a specific target. This is considered a violation of the self-organisation principle, as a global update algorithm (i.e. the optimizer) adjusts all network weights to minimise a global target function.

Claim 1 *End-to-end backpropagation of error violates the principle of self-organization. Self-organisation in neural networks requires dividing the network into smaller units that are optimised independently of each other.*

In fact, the plausibility of backpropagation of error for explaining how the brain works was questioned soon after it was published [137, 138]. Since then, many alternative and biologically more plausible algorithms have been proposed such as the feedback alignment (FA) algorithm [240], generalized recirculation [241], as well as target propagation (TP) [242]

[137]: Crick (1989)

[138]: Grossberg (1987)

[240]: Lillicrap et al. (2014)

[241]: O'Reilly (1996)

[242]: Le Cun (1986)

(c.f. Section 3.3). However, Bartunov et al. [168] have demonstrated that these algorithms do not scale to large vision datasets such as ImageNet [243] and only work for smaller datasets such as MNIST [244] and CIFAR-10 [245]. The only algorithm that seems to scale well is using a proxy objective functions (c.f. Section 3.3).

[168]: Bartunov et al. (2

The biologically most plausible learning algorithm is Hebbian learning (c.f. Section Section 2.5.1) and its variants such as contrastive Hebbian learning [246]. However, even tough I obtain some promising results in preliminary experiments with Hebbian learning (c.f. Appendix Chapter A), this algorithm doesn't seem to be well suited to learn good image representation if a network is trained from scratch.

[246]: Movellan (1991)

Thus, the use of proxy objective functions[1] seems promising; the updates of weights are done in separate local units, yet the power of current deep learning training algorithms can be exploited and systems can be created that can solve complex problems and scale to large datasets by interacting with each other.

1: proxy objective functions are loss functions that are only applied to local units of a system

Implementation 1.1 *Instead of using end-to-end backpropagation of error to optimise the whole system according to a single global objective function, proxy objective functions are applied to local units. Thus, self-organisation takes place through the optimisation of local units instead of an overall system.*

The next ambiguity is what local units are in a deep learning setting. In deep learning models, typically not neurons are modelled but the trainable parameters[2]. One of the strengths of deep learning systems is that matrix multiplications allow to calculate the layer outputs in one step. Calculating each neuron activity separately, on the other hand, would be very inefficient. Therefore, the smallest meaningful unit for local updates seems to be a layer and not a single neuron. If a neural network is visualised layer-wise from left to right, then the self-organising units line up horizontally. This is why layer-wise self-organisation is also referred to as "horizontal self-organisation" in this thesis (c.f. Figure 4.1).

2: the weights w and the bias b are modelled, so that the output y can be calculated as $y = w \cdot x + b$ for given data x

Implementation 1.2 *Self-organisation takes place within local units. A local unit can be a part of a model such as layer. In this thesis, this type of self-organisation is called horizontal self-organisation.*

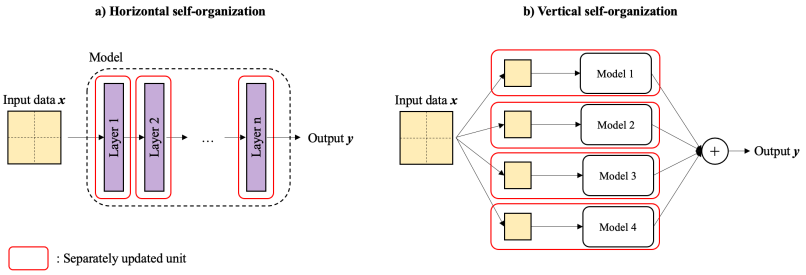

Figure 4.1: Two different ways of building self-organizing units. Self-organization can either take place horizontally (i.e. layer-wise) within a model (a) or vertically by splitting the data into patches and processing them with independent units (b). The independent units are marked with a red frame.

A second type of self-organisation is not to split the model into separate units but to split the data. The input data can be divided into smaller patches and then be processed by independent models. It is important that one model does not process the entire set of existing patches, as is the case with the vision transformer. If only one model is used, there would again be an end-to-end backpropagation of error on a single unit.

But if the patches are processed by a graph of independent models, then each model can be considered as a self-organising unit. In this thesis we call this kind of self-organisation the "vertical self-organisation" (c.f. Figure 4.1).

Implementation 1.3 *A second type of self-organising unit can be a model that processes a subset of input data that is not shared with other models. In this thesis, this type of self-organisation is called vertical self-organisation.*

4.1.2 Net-Fragments

Another very important principle according to von der Malsburg et al. [2] is that neurons form net-fragments (a.k.a. sub-networks) that represent features of objects (c.f. section Section 3.1). For example, some net-fragments may represent shapes and structures while a multitude of such net-fragments together represent objects such as persons or entire scenes. Net fragments are a compositional data structure, meaning that some low-level features can be composed to a higher-level feature and multiple higher-level features are composed to an object. To some extend, neural networks do this as well; data is fed into the network, the first layer extract some low-level patterns, and subsequent layer combine these patterns in higher-level features in a hierarchical manner. However, there is one big difference: The features that are used to fulfil a specific task such as classification are extracted from the latent space of *one* single layer. Net-fragments in the human brain, on the other hand, are not considered to be present at a specific point in time but to be built up over a short period of time.

Claim 2 *Net fragments are represented by groups of neurons and their activity over multiple time-dependent spikes. In addition, several net fragments can be composed into a higher-level fragments. Thus, objects are not represented by a single neuron but by all neurons that were activated due to this object.*

Deep learning models are not based on time-dependent spikes of neurons that compose features to more complex features. However, a time-step could be interpreted as a forward-step from one layer to the next within a network of layers. Thus, net-fragments would be the activation of multiple layers. This interpretation is also in line with the compositional property of net-fragments. Low-level features that are detect within the first layers activate higher-level features in subsequent layers. An object, however, is not represented in the last layer of a neural network but through all activations within the network. Thus, representations of objects cannot be extracted from a single layer but from multiple layers. Intuitively, this seems promising. For example, auto-regressive models applied on speech capture different information at different layers of the network [247]. While the first layers contain more information to distinguish speakers, representations in later layers provide more phonetic content. Thus, extracting information from several layers could lead to representations containing more information.

[247]: Chung et al. (2019)

Implementation 2.1 *Net-fragments cannot be extracted from one single layer. Therefore, the representations of an image are extracted from multiple layers.*

In the human brain, distinct groups of neurons represent specific net fragments. This means that neurons are only active when the corresponding feature is present in the input and are inactive otherwise. Moreover, the features represented by the neurons should be meaningful[3] and consequently not active for every existing object. This inevitably leads to the sparse and diverse activations. The activations are sparse and diverse because, the object in the image consists of only a small sub-set of all learned features. Thus, only a small sub-set of the neurons should be active for a given object. When the input changes and a different object is shown to the model, also the set of active neurons should change.

3: interpretable in the sense that a neuron is active if a specific feature is present and inactive otherwise

Claim 3 *The activation patterns of neurons that represent net-fragments must be sparse and diverse to obtain meaningful activation patterns.*

The extent to which neuronal networks contain or can produce net fragments with meaningful activation patterns is investigated in detail in a preliminary study. The methodology used as well as an in-depth evaluation is described in the appendix in the Chapter Chapter B. In summary, it was found that neural networks do not contain net fragments with meaningful activation patterns by default. Typically, there are neurons that are always active regardless of the input data and encode most of the input information in their activation strength. Other neurons, however, are never active and are therefore not needed by the network. If, on the other hand, a sparsity and diversity constraint is applied, then layers are obtained with neurons that appear to be suitable for net fragments.

Implementation 3.1 *Sparse and diverse activation patterns can be obtained by imposing sparsity and diversity constraints to the objective function of the model.*

However, sparsity and diversity constraints are mainly necessary to obtain meaningful and more robust activation patterns. They don't seem to be necessary, for example, if the goal is to obtain good classification performance. An alternative that lead also to robust and meaningful activations is to model the net-fragments as probability distribution. For example, variational autoencoders (c.f. Section 3.4) model their latent space as a multivariate Gaussian distribution.

Sparsity

Sparistiy seems to be an important principle in biological networks. Presumably, this is because the neurons in the brain can be inhibitory or excitatory by firing spikes at different time intervals. An artificial neuron, on the other hand, uses a floating point number as its state and can thus represent an infinite number of different states, in contrast to the biological neuron. Thus, an artificial neuron is able to represent many different things while the biological neuron is responsible for specific features. Therefore, the activations of biological neurons *must be* sparse, while the activations of artificial neurons *do not have to be* sparse (but being sparse has also advantages for artificial neurons, see below).

Sparsity is deeply embedded in the biological learning process. For example, in the visual cortex of mice are more than 75% of the neurons

active before the first opening of the eyes, 36% after the opening of the eyes and only 12% in adulthood [248]. Thus, a sparsification of neuronal activations takes place through visual experience. In the field of deep learning, sparsity is often interpreted in two different forms; sparse weight matrices and sparse activation matrices. Sparse weight matrices are often chosen to make models smaller or to increase inference speed [249, 250]. From a biological point of view, this process of first creating a large network and then shrinking it is obviously not plausible[4]. Sparse activations, on the other hand, can increase robustness [251]. Intuitively, sparse activations enforce that only the most relevant information is passed to the subsequent layer. Furthermore, combining sparsity with diversity can help to obtain more interpretable activations (c.f. Appendix Chapter B).

[248]: Rochefort et al. (2009)

[249]: Louizos et al. (2018)

[250]: Hoefler et al. (2021)

4: otherwise we would have a large brain at the beginning, which becomes smaller by factors in the course of time

4.1.3 Lateral Connections

In addition to forward connections, lateral connections are also located in visual cortex [252]. Thus, the biological neurons are not only connected to the neurons in the subsequent layer but also within the same layer. Von der Malsburg et al. [2] describe that lateral connections help active neurons to support each other in order to remain active: Initially, many neurons are active, but relatively quickly a part of the neuron becomes inactive and only those neurons that support themselves remain active. In this way, lateral connections can lead to the emergence of high-level features from initial activations.

[252]: Gilbert et al. (1990)

Claim 4 *Lateral connections allow active neurons to support each other and to remain active.*

Neural networks lack this time dynamic and neurons do not suddenly become inactive during a forward pass. In addition, initial experiments have shown that performance does not improve when the activation maps of neural networks are sparsified over several steps. One assumption is that all information is already contained in the data at the beginning and the same sparsified activations can also be generated in one initial step.

However, the lateral support between neurons seems promising if the input changes slightly: An activation map can be calculated for a given input image. In a next step, the input can slightly be changed through data augmentation to btain another activation map from the same image. Thus, the network receives different views of the same image and generates different activation maps of the same image. In every step, the model can decide whether the activation map from the previous step was correct or if the activation map should be updated.

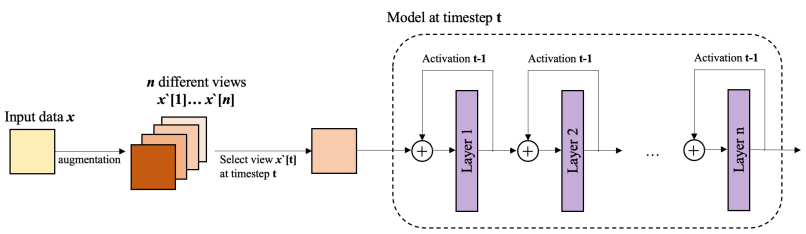

Figure 4.2: An illustrative network architecture of a model with lateral connections: An in sample x is augmented n times to obtain n different views $x'[1], ..., x'[n]$ of the same sample. At every timestep t, the sample $x'[t]$ is fed through the model. Each layer receives the activation map of the previous layer as input as well as its own activation maps from the previous time-step $t-1$.

This could be implemented by interpreting the lateral connection as a recurrent connection. An illustrative architecture of such a model is shown in Figure Figure 4.2. A data sample x can be augmented with data augmentation methods n times to obtain n different views $x'[1], ..., x'[n]$ of the input sample. Afterwards, the n views are iteratively fed into the model. At every time-step t, the model calculates an activation map $a_l[t]$ for each layer l. If $t > 1$, the input of the layer l is not only the activation map of the layer $l-1$ but also the activation map of the previous time-step $a_l[t-1]$. The activation map of the previous time-step is stored $a_l[t-1]$ by the recurrent (lateral) connection. Thus, the activation map $a_l[t]$ is not only calculated based on the activation map of the previous layer $a_{l-1}[t]$ but also based on the activation map of the previous time-step $a_l[t-1]$. This allows each layer to preserve the activation map $a_l[t-1]$ from the previous time-step or to correct it based on a slightly different input $a_{l-1}[t]$ from the previous layer. The lateral connections thus support the activations within a layer; a layer can look at several inputs and decide which features are present over several time-steps (support these features over several time-steps).

Implementation 4.1 *A lateral connection can be implemented as a recurrent connection that "stores" the layer's activation of a previous time-step. If the same input is present for several time-steps in a slightly augmented version, the layers can keep their previous activation maps (lateral support) or correct them.*

With vertical self-organisation, the lateral connections can be implemented not only as recurrent connections within layers, but also as connections between the sub-models. In this case, each sub-model extracts a distinct low-level feature. Based on this feature, a guess can be made to which higher-level feature or object the extracted feature belongs to. Through communication with neighbouring sub-models, this guess can either be supported or rejected. Thus, the lateral connections are used as a "support channel" between sub-models.

Implementation 4.2 *In the case of vertical self-organisation, a lateral connection can be implemented as connection between sub-models to vote for which high-level feature or object is represented in the image. Either the prediction of a model can be supported by neighbouring models or rejected.*

4.1.4 Other Principles

Other important principles that are considered promising are a *continuous input* signal and an *embodiment* of the agent. However, implementing such an interaction between an agent and an outside world is out of scope for this thesis but might be interesting for future work.

The visual cortex receives a continuous input signal. This allows the tracking of moving objects and enables an object to be perceived from different angles. Since the change of the object between the captured frames is small, it can be determined that it is always the same object instance and consequently mapped to the same mental object prototype of a world model.

An ANN, on the other hand, is typically trained on samples that have little relation to each other. When the system is trained on images, each

frame is different; with videos, each sequence of frames is different. A continuous input might help to get better representation of objects through self-supervised learning. If an input is continuous and shows the same object from different angles or in different transformations (e.g. stretching) and it can be inferred that it is the same object then the object representations derived from this continuous stream can be homogenized. These principles are already applied to some extent by self-supervised learning systems for computer vision. In contrastive learning, a popular form of self-supervised learning, two different views are derived from one image by data augmentation, and their representations are then pushed closed together in the feature space [187, 189, 253]. However, this paradigm is still quite limited since only two views of the same scene and not the continuous transformation of an object are presented to the learning system.

[187]: Chen et al. (2020)

[253]: Chen et al. (2020)

[189]: Caron et al. (2020)

Claim 5 *A continuous input stream can help to build a better representation of objects, especially if the objects are slightly transformed between captured frames or if the point of view changes continuously and smoothly.*

Furthermore, efference copies of motor signals in the form of neuronal activities are directly sent to the brain's sensory system if animals are moving [254]. Such efference copies can be useful to better understand objects and how they behave when undergoing object transformations. To do so, the agent must be allowed to perform actions and interact with its world. This gives the agent more information about the objects but also about the physical properties of a (virtual or real) world. For example, he can perform different actions on different objects: He can rotate, squeeze, stretch or move objects. By doing so, he receives different visual and sensory feedback for identical actions on different objects. The agent can map the captured feedback signals to the object representations of an internal world model.

[254]: Keller et al. (2012)

Claim 6 *Allowing an agent to interact with the world can help to learn better object representations and to build a better world model.*

Furthermore, the agent can learn to understand objects better by improving his internal object representations on its own. For example, if the agent examines an object and his current internal object representation does not describe how the object looks from the side, then the agent can rotate the object accordingly and complete or correct the representation. Such a behaviour could be implemented within the agent itself, for example, by optimising an entropy-based loss in order that the agent explores the world and tries to learn unknown things [255, 256].

[255]: Storck et al. (1995)

[256]: Klapper-Rybicka et al. (2001)

4.2 Vertical Self-Organization

In this thesis, two different concepts of self-organisation are implemented. The first uses vertical self-organisation. This means that within a model, smaller units are identified that are trained independently of the other units. (c.f. Section 4.1.1). A linear layer was selected as one unit. This is a small unit (e.g. compared to a model part that contains many layers) but can still be trained efficiently. Smaller units would be neurons or parts

of a layer, but training them separately would be very inefficient, since the efficiency of matrix operations on GPUs could no longer be fully exploited[5].

5: the layer output can be efficiently calculated by a single matrix multiplications and addition, i.e. $z = w \cdot x + b$ (c.f. Section 2.2)

Each layer optimises its own proxy objective function. Proxy objective functions are used because this type of local optimisation allows very good performance even on larger data sets, in contrast to other target functions such as target propagation, synthetic gradients or feedback alignment (c.f. Section 3.3). Figure Figure 4.3 visualizes the updates based on a proxy objective function. A objective function is calculated for each layer and the optimization algorithm only updates the parameters within this layer. Thus, the gradients do not flow backwards to preceding layers.

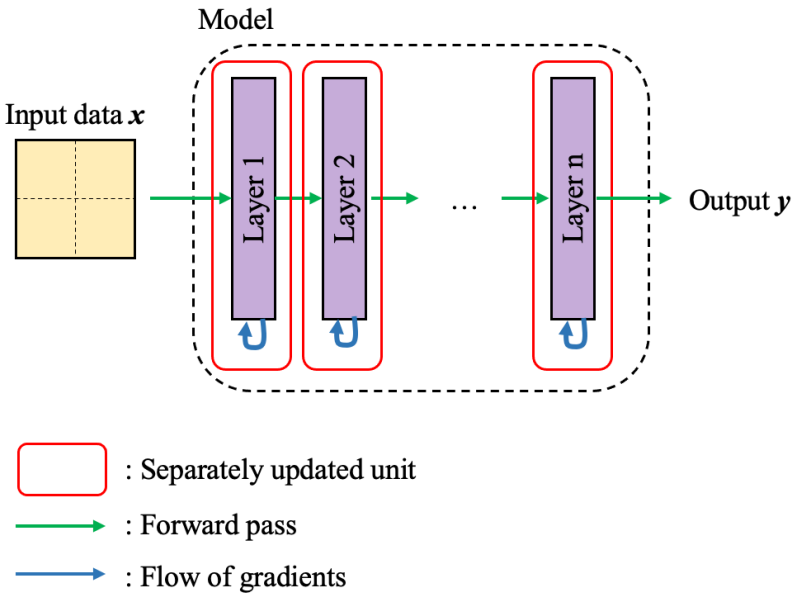

Figure 4.3: The flow of gradients within the network based on vertical self-organisation: The data is fed from layer to layer during the forward pass. The layers, are trained independently and the gradients do not flow from one layer to the previous one.

A combination of diversity and sparsity constraints is used as the loss function, which leads to representations that are easy to interpret and suitable for net fragments (c.f. Section 4.1.2) and also increases robustness (c.f. Section 3). Identical to the preliminary experiments in Appendix Chapter B, the sparsity is achieved by using the kullback-leibler (KL) divergence [177]. The model consists of several linear layers l with a relu activation function $(z)^+ = max(0, z)$. A layer consists of $m^{(l)}$ neurons and the activations $z^{(l)}$ of a linear layer l are calculated for a given input $z^{(l-1)}$ as

[177]: Le et al. (2012)

$$z^{(l)} = z_1^{(l)}, ..., z_m^{(l)} = w^{(l)} \cdot z^{(l-1)} + b^{(l)} \tag{4.1}$$

whereby $w^{(l)}$ is the weight and $b^{(l)}$ the bias of layer l.

The activation probability can be calculated for each neuron. If a mini-batch contains n samples, the activation probability $\hat{\rho}^{(i)}$ of a neuron $z^{(i)}$ can be calculated as:

$$\hat{\rho}_i^{(l)} = \frac{1}{n} \sum_i^n \frac{1}{1 + e^{-z_i^{(l)}}} \tag{4.2}$$

where $\frac{1}{1+e^{-z^{(i)}}}$ is the sigmoid function that squeezes the activation in the range between 0 and 1. With the KL divergence, the divergence of the current activation probability $\hat{\rho}^{(i)}$ and a desired activation probability $\rho = 0.05$ can be calculated:

$$KL(\rho||\hat{\rho}_i^{(l)}) = \rho \cdot \log \frac{\rho}{\hat{\rho}_i^{(l)}} + (1-\rho) \cdot \log \frac{1-\rho}{1-\hat{\rho}_i^{(l)}} \tag{4.3}$$

The sparsity loss L_s is the sum of the divergence between all $\hat{\rho}^{(i)}$ and ρ:

$$L_s(\rho, \hat{\rho}) = \sum_{i=1}^{m} KL(\rho||\hat{\rho}_i^{(l)}) \tag{4.4}$$

The second constraint is a diversity constraint. The goal is that the activations of *different* objects are diverse. For this purpose, the activations $\boldsymbol{z}^{(l)}$ are made as identical as possible (i.e. pushed together in feature space) if they stem from the same class and as different as possible if they stem from different classes. The cosine similarity is used to calculate the similarity between two activations $z_i^{(l)}$ and $z_j^{(l)}$:

$$\cos(z_i^{(l)}, z_j^{(l)}) = \frac{z_i^{(l)} \cdot z_j^{(l)}}{\max(||z_i^{(l)}||_2, ||z_j^{(l)}||_2)} \tag{4.5}$$

In order to make representations of different objects different, resp. to make representations of identical objects identical, the information of image labels y_i is needed. Thus, $y_i \in C$ where C is the set of classes. The diversity loss L_d minimises the similarity $\cos(z_i^{(l)}, z_j^{(l)})$ if the two activations stem from different classes $y_i \neq y_j$, or maximises the similarity if they stem from the same class $y_i = y_j$.

$$L_d(\boldsymbol{z}^{(l)}) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k \cdot \cos(z_i^{(l)}, z_j^{(l)}) \tag{4.6}$$

whereby k changes sign depending on the class:

$$k = \begin{cases} +1, & \text{if } y_i \neq y_j \\ -1, & \text{otherwise} \end{cases} \tag{4.7}$$

However, it was found that the loss is more stable if the similarity is not calculated between two activations but between one activation and the average activation of a class. The average activation of a class c can be calculated over n_c samples from this class as:

$$z^{(l)}[c] = \frac{1}{n} \sum_{i=1}^{n_c} z_i^{(l)}, \text{ for } y_i = c \tag{4.8}$$

Another problem of is loss is if all activations are identical, then $\cos(z_i^{(l)}, z_j^{(l)}) = 1$ and the loss gets $L_d = 0$, since the similarities of the same and different

[257]: Balntas et al. (2016)

classes neutralise each other. Therefore, a margin between the similarities is enforced, as done for the triplet-margin-loss [257].

$$L_d(z^{(l)}) = \frac{1}{n}\sum_{i=1}^{n} \max(\cos(z_i^{(l)}, z^{(l)}[v]) - \cos(z_i^{(l)}, z^{(l)}[y_i]) + \text{margin}, 1) \tag{4.9}$$

where v is a random class drawn from the set $v \{C$
$y_i\}$ and margin is a hpyer-parameter that was set to 1. Thus, the triplet-margin-loss is calculated but the $L2$-norm is replaced by the cosine-similarity as distance measure and the positive, resp. negative anchor is replaced with the average class activation from the same resp. from a randomly selected different class.

The loss used is the sum of sparsity loss and diversity loss, with the diversity loss weighted by $\lambda = 0.1$:

$$L = L_s(\rho, \hat{\rho}) + \lambda \cdot L_d(z^{(l)}) \tag{4.10}$$

One problem with proxy objective functions is that this type of training often requires layer-wise training. First, layer 1 is trained completely, then the weights are frozen, then layer 2 is trained and so on. This is inefficient because it requires more forward-passes than if the model is trained with end-to-end backpropagation. It was found that this loss functions allows to train all layers simultaneously. With a forward-pass, all activations are calculated, followed by a layer-wise backward-pass where the gradients from one layer do not propagate back into the previous layer. Since the gradients are only needed locally, no large graph of gradients has to be calculated, which reduces the memory utilization on GPUs and allows larger mini-batch sizes. In addition, this type of architecture is very easy to parallelise, since theoretically each layer (i.e. each independently trained unit) or a group of layers can be assigned to a different GPU. In contrast to end-to-end backpropagation of error architectures, the gradients only flow backwards locally on a GPU and do not have to be passed on to other GPUs.

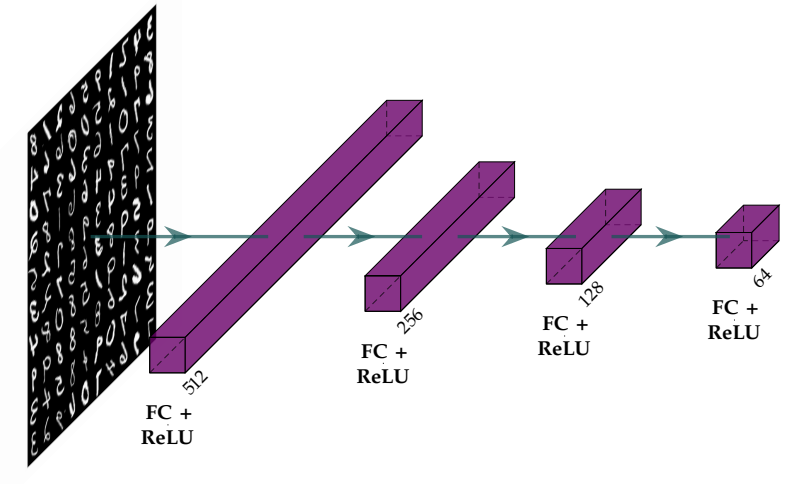

Figure 4.4: The network architecture of the fully connected model for vertical self-organisation with fully connected layers.

The model used consists of 4 fully connected layers with relu activation. The first layer has 512 neurons, the second 256 neurons, the third 128 neurons and the fourth 64 neurons. The model is illustrated in Figure Figure 4.4. Each layer is trained separately by minimising the loss of equation equation (**??**) with the Adam optimizer[258] and a learning rate of $\eta = 1 \cdot 10^{-3}$. The mini-batch size is 60, 000.

[258]: Kingma et al. (2017)

4.2.1 Extraction of Representations

In accordance with net-fragments as in Section 4.1.2, the representations are not extracted at a specific point (i.e. a pre-defined layer) but all representations from all layers are taken into account to fulfil a task. In the following, this is demonstrated based on a classification task, but other tasks are also conceivable in the future. After training, the average activation $z^{(l)}[c]$ for each class $c \in C$ in each layer l is determined, as done in Equation equation (**??**). These averages from the training set represent prototypes of each class object in each layer and can be considered as reference representation per class. Thus, the representations needed for this task are computed *after* training and are not part of the training as for example in the case of a classification loss based on cross-entropy[6]. When a new sample $\boldsymbol{x}_s$ is classified, the cosine similarity between the activations $\boldsymbol{z}_s^{(l)}$ of this sample and the class prototypes $\boldsymbol{z}^{(l)}[c]$ is calculated in each layer.

6: but in the case of classification this information is implicit in the loss function

$$\cos_s^{(l)}[c] = \cos(z\boldsymbol{z}_s^{(l)}, \boldsymbol{z}^{(l)}[c]) = \frac{\boldsymbol{z}_s^{(l)} \cdot \boldsymbol{z}^{(l)}[c]}{\max(||\boldsymbol{z}_s^{(l)}||_2, ||\boldsymbol{z}^{(l)}[c]||_2)}, \text{for } c \in C \tag{4.11}$$

Thus, the cosine similarity $\cos^{(l)}[c]$ for each class $c \in C$ and each of the for layers $l \in 1, ..., 4$ is calculated. Afterwards, the average class c with the highest average cosine similarity between the sample activations $\boldsymbol{z}_s^{(l)}$ and the class prototypes $\boldsymbol{z}^{(l)}[c]$ is used as prediction.

$$\arg\max_{c \in C} \frac{1}{4} \sum_{l=1}^{4} \cos_s^{(l)}[c] \tag{4.12}$$

This results in a weighted voting; if a layer is very sure that the sample belongs to a specific class, then the sample has a high cosine similarity with one class prototype and a low similarity with all other class prototypes. Accordingly, this layer influences the prediction more than a layer that cannot clearly assign the sample to one class and calculates a similarly high cosine similarity between the sample and all prototypes.

4.2.2 Lateral Connections

As described in section Section 4.1.3, lateral connections serve neurons to support their activations among each other. In this chapter it is also described in detail that this can be implemented through recurrent connections. There are several ways to implement recurrent connections. A very simple possibility is to concatenate the layer input $\boldsymbol{z}_t^{(l-1)}$ at time

t with the layer output $z_{t-1}^{(l)}$ at time $t-1$. This is visualized in Figure Figure 4.5. Of course, $z_{t-1}^{(l)}$ is undefined at $t = 0$. In this case, $z_{t-1}^{(l)}$ is initialized with zeros.

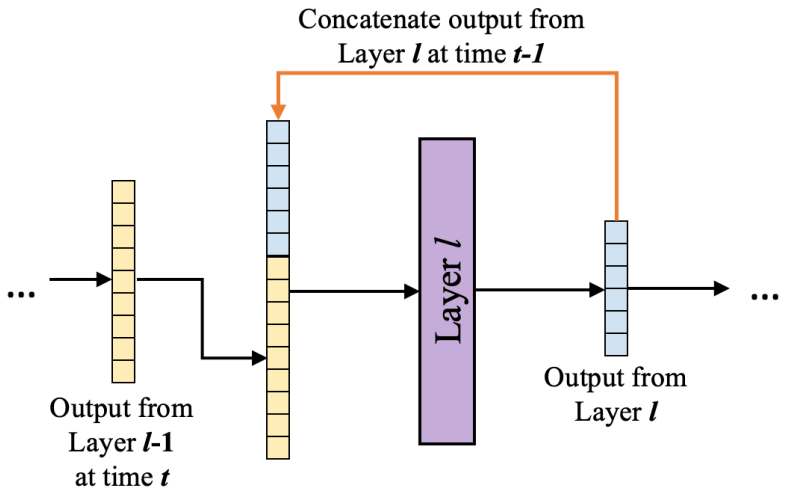

Figure 4.5: The lateral connections can be implemented by concatenating the layer's output at the previous time-step with the layer's input at the current time-step.

A second option is to use a second weight matrix as it is usually done in recurrent layers and to expand the equation equation (**??**) for the activation function as follows:

$$z_t^{(l)} = \boldsymbol{w}_x^{(l)} \cdot \boldsymbol{z}_t^{(l-1)} + \boldsymbol{b}^{(l)} + overbrace\boldsymbol{w}_h^{(l)} \cdot \boldsymbol{z}_{t-1}^{(l)}{}^{\text{lateral connection}} \tag{4.13}$$

whereby $\boldsymbol{w}_x^{(l)}$ is the weight multiplied with the layer input and $\boldsymbol{w}_h^{(l)}$ the weight multiplied with the previous layer output. In both cases, the layer receives information about the activations at the previous time-step. However, this only seems helpful if the model input is not static. Therefore, the model input is available over several time-steps and is augmented after each time-step. Thus, the model receives different views of the same image and can adjust its activations from the previous time-step if necessary. The following image augmentation techniques are applied, each with a probability of $p = 0.8$:

- **Color Jitter**: Randomly change brightness, contrast, and saturation of the image.
- **Gaussian Blur**: Blur the image with randomly chosen Gaussian blur.
- **Random Rotation**: Randomly rotate the image with an angle in the range $[-15, ..., 15]$
- **Adjust Sharpness**: Randomly adjust the sharpness of the image.

Figure Figure 4.6 visualises how this augmentation affects samples of the MNIST data set [16]. The original image is shown on the left and 9 augmented versions of it are shown on the right. This allows the model to perceive the same image from different views.

Recurrent connections, which in this context represent lateral connections, are typically used to process sequential data or text. In this case, the data

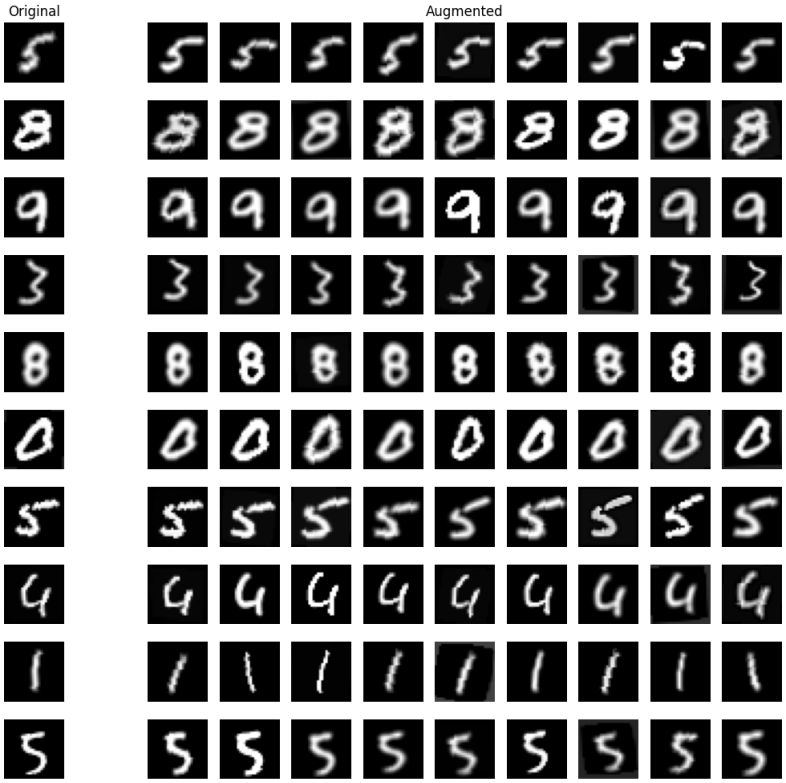

Figure 4.6: Data augmentation applied on 10 samples of the MNIST data set. The original samples as it is in the data set is shown on the left, 9 augmented versions of the same samples are shown on the right.

is collected cumulatively before an output is generated. For example, in text processing, all word tokens of a sentence are typically read before the model classifies the sentence as heat-speech or not. This is necessary because all sentence information is required and classification cannot be done on the basis of a single token. Such models are typically trained with backpropagation through time (BTT). Thereby, the gradients flow backwards over several time steps. This leads to well-known problems such as vanishing and exploding gradients.

In this thesis, BTT is not used, which means that a prediction is made after each time-step, but the prediction potentially improves with more time-steps. This leads to desirable properties: (i) Problems with vanishing or exploiding gradients do not exist, regardless of how many time-steps the model requires. (ii) After each time-step, representations can be extracted according to Section Section 4.2.1. If a task can be solved correctly with a high probability on the basis of these representations (e.g. the object representation can clearly be assigned to one class label), the sequential analysis of the image can be aborted. If this is not the case, further time-steps can be carried out until the model has a sufficiently high confidence in its prediction. Thus, the number of time-steps can be sample-dependent.

4.2.3 Hierarchical Features

A criticism of the proposed model is that it does not learn hierarchical features, even tough hierarchical features are one of the main reasons for the good performance of deep learning systems. The diversity loss (c.f. Equation equation (**??**)) forces enforces in each layer that the latent

representations of objects of the same class are similar and that the representations of different classes are different. This violates the concept of hierarchical feautres; the first layers should learn general features that are helpful for all classes, but cannot necessarily be assigned to a specific class. Only later layers build class representations that are specific to a class. In the current setting, however, already the first layers generate class specific representations.

In the following, three possible measures are described to counteract this problem: (i) The fully connected layers are replaced by convolutional layers, so that the first layers have a smaller field of view and can only recognise local features. (ii) An adapted version of diversity loss forces a separation of features by class only in the last layers. (iii) While the very specific image information flows into the network from one side, general information in the form of class labels is fed into the network from the other side and these two types of information are fused. These three measures can be applied either individually or in combination with each other.

Convolutional Architecture

corresponding layer has a sufficiently large field-of-view[7]. In a CNN, the field-of-view in the first layers is restricted by design, but not in fully connected layers. Consequently, it might help to use a CNN architecture instead of a model based on fully connected layers only. This means that the network is no longer able to separate the representations based on the class in the first layers as the field-of-view is too small, but can only do this in the later layers, which have a larger field-of-view.

7: the field-of-view are all pixels of the input image that can influence a neuron's activity

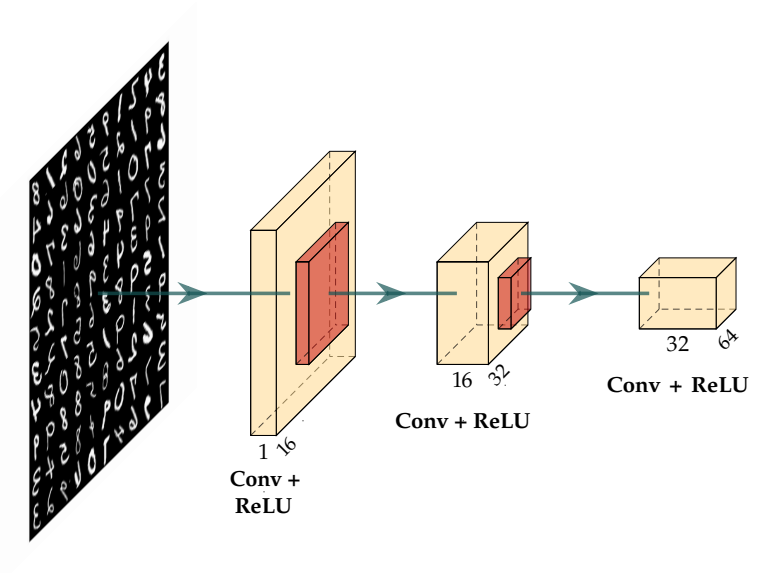

Figure 4.7: The network architecture of the CNN for vertical self-organisation with fully connected layers.

The CNN architecture used in this thesis is shown in Figure Figure 4.7. Three convolutional layers with ReLU activation function and with 16, 32, and 64 channels are used, and max-pooling layers are applied between each convolutional layer. For training, the same hyperparameters and loss functions are used as for the model with fully connected layers.

Hierarchical Diversity Loss

A second measure can be to adapt the diversity constraint of the loss function. In the current version, it forces the latent representations of objects of the same class to be similar and those of different classes to be different. This is useful in the last layers, where high-level features (i.e. high-level net-fragments) or object representations should be detected and be separated from each other. In the first layers, on the other hand, the separation should not depend on the class label. Nevertheless, the activations should also be diverse if low-level features are detected (c.f. Section Section 4.1.2).

Therefore, the sparsity constraint is split into two parts: One part ensures that the activations within a (large) mini-batch are diverse and thus enforces that different features are captured and represented by different neurons. The second part ensures, as before, that the activations are diverse for different classes. Thus, one part of the constraint ensures *diversity within the mini-batch* and the other part ensures *diversity between different classes*.

These two parts are weighted linearly from the first to the last layer. Since the first layer should have a high diversity within the mini-batch, it has a high weight on the first part of the diversity constraint and a low weight on the second part. The last layer has an inverse weighting and pays more attention to the second part of the diversity constraint than to the first part.

The first part of the diversity constraints ensures diversity within a mini-batch. This is achieved by ensuring that each neuron within a mini-batch should be active.

TODO: At the moment, various experiments are still ongoing to find out how this can be implemented (therefore, not yet explained in more detail).

Two-Way Information Flow

Hinton [156] introduced with the forward-forward (FF) algorithm (c.f. Section Section 3.3) a promising idea for models based on proxy objective functions; The image is fed into the input layer of the network and the corresponding label is fed into the output layer of the network. The image remains static for several time-steps and the network is trained to push the layer's activations above a certain threshold. In a second stage, the image is fed into the network together with the wrong label and the network is trained to push the activations below a certain threshold. He found that when low-level features (i.e. images) are fed into the network from one end and high-level features (i.e. class labels) are fed into the network at the other end, a feature hierarchy is created. However, this approach has one major disadvantage: During inference, each possible sample-label combination has to be fed into the model and the label that caused the highest neural activity is used as the model's prediction.

[156]: Hinton (2022)

In this thesis, this is implemented in a different way, which does not have this disadvantage. Identical to the FF algorithm, the image is fed into the input layer of the network for multiple time-steps. The label, on the other

hand, is not fed directly into the output layer but is made available to the last layer within the loss function. Each layer maximizes the mutual information (MI) between its own activations and the activations of the previous and subsequent layers. Thus, the first layer maximizes the MI between the input image and its activations, the last layer maximises the MI between its activations and a label vector. This creates a feature pyramid in which a smooth transition from concrete sample to abstract class label is learned. During inference, the last label then directly predicts latent representations that can be assigned to a class.

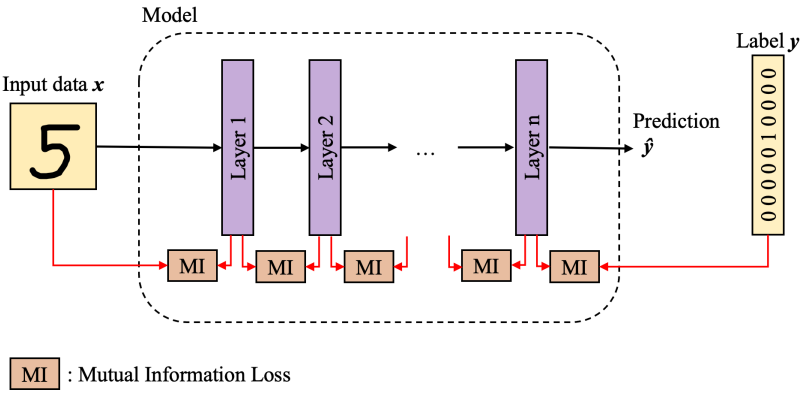

Figure 4.8: Vertical self-organization with mutual information loss: Each layer maximizes the mutual information (MI) between its activations and the activations of the previous and subsequent layer. The first layer maximizes the MI between its activations and the input image (instead of the previous layer) and the last layer maximizes the MI between its activations and the image label (instead of the subsequent layer).

This process is visualized in Figure Figure 4.8.

TODO: At the moment, various experiments are still ongoing to find out how this can be implemented (therefore, not yet explained in more detail).

4.3 Horizontal Self-Organization

Results 5

6 Future Work

6.1 Useful Representations

In this thesis I describe an approach to generate representations of a visual scene. However, the question how useful this representations are still remains[1]. The ultimate goal of perception systems is to build a simple and informative model of the external world. This internal model should not be too detailed but include all behavior-relevant information. Thus, the internal model should allow an agent to take the optimal action to fulfil a given task.

1: despite for the usual ML tasks such as image classifications

It is known from the study of animals that both eye movements and the behavioral state influence the responses of neurons in the visual cortex [254]. Thus, animals integrate their action (i.e. the movement they are doing) with currently incoming sensory signals to predict future sensory inputs. The internal copy of an outflowing movement-producing signal generated by an organism's motor system is also known as efference copy. Keurti et al. [259] argue that such efference copies are useful to learn *useful* latent representations perceived by the visual system. They translated this idea into an AI-based system by allowing an agent to interact with the environment and to observe its state to build internal representations.

[254]: Keller et al. (2012)

[259]: Keurti et al. (2022)

If useful latent representations of a visual scene are considered to be a world-model with all behavior-relevant information, it should be evaluated whether the representations obtain by the proposed model correspond to this definition. If not, which is likely according to the author's intuition, the latent representations should be optimized accordingly. This implies that the existing perception system should be extended by cognition. Thus, the AI system needs an embodiment to interact with the environment. For example, this can be simulated with a reinforcement learning based agent. The training process could be explicitly modeled by predicting future states based on a given state and possible actions before the action is executed and the actual outcome in the world model is observed. This procedure corresponds to the perception-action episode that was proposed by LeCun [260]. He divides the process in seven steps; (i) First, the perception system extracts a representation of the current state of the world $s[0] = P(x)$. (ii) The actor then proposes an initial sequence of actions $(a[0], ..., a[t], ...a[T])$ that is evaluated by the world model. (iii) The world model in turn predicts likely sequence of world state representations resulting from the proposed action sequence $(s[1], ..., s[t], ...s[T])$. (iv) A cost model estimates the total costs for each state sequence as a sum over time steps $F(x) = \sum_{t=1}^{T} C(s[t])$. (v) Based on the cost predictions, the actor proposes the action sequence with the lowest costs. (vi) The actor then executes one or a few actions (and not the entire action sequence) and the entire process is repeated. (vii) Additionally, every action, the states and associated costs are stored in a short-term memory that can be used to optimize the system.

[260]: LeCun (2022)

Appendix

Experiments Hebbian Learning

A

I used different variants of Hebbian Learning with the goal of generating good latent representations of visual scenes. However, all preliminary experiments with different implementations were not promising. In my experiments, classical Hebbian learning (as described in Equation equation (**??**)) led to symmetric weights, which in turn led to poor representations. This symmetry could be broken by using the ABCD-Hebbian learning rule [261]. The ABCD learning rule calculates the weight update Δw_{ij} from neuron i to neuron j based on the pre-synaptic activity r_i of neuron i and post-synaptic activity r_j of neuron j as follows:

[261]: Niv et al. (2001)

$$\Delta w_{ij} = \eta_w \cdot (A_w r_i r_j + B_w r_i + C_w r_j + D_w) \tag{A.1}$$

where η_w is a weight-specific learning rate, and A_w is a correlation coefficient, B_w is a presynaptic coefficient, C_w is a postsynaptic coefficient, and D_w is a bias coefficient. The bias coefficient D_w can be interpreted as an individual inhibitory or excitatory bias of each connection in the network. Similar to Najarro and Risi [111], the coefficient were learnt through an evolution strategy [235]. However, this method does not scale for large networks with many parameters and did not achieve the desired performance in my experiments.

[111]: Najarro et al. (2020)

[235]: Salimans et al. (2017)

Of course, it cannot be concluded from these experiments that Hebbian Learning cannot be used to learn networks for extracting good representation from visual scenes. However, it was found that it is not trivial and that just applying the Hebbian learning rule is not sufficient, especially if the network has a structure of modern Deep Learning architectures with many parameters.

However, in further preliminary experiments it was found that Hebbian Learning has interesting properties for pruning. A good performing CNN was created and trained on MNIST with backpropagation. The network consists of two convolutional layers, followed by a pooling layer and three linear layers. The first convolutional layer increases the number of channels from [width × height × n_channels] to [width × height × 32]. The second convolutional layer further increases the number of channels to [width × height × 64]. The subsequent max. pooling layer decreases the size to [width/2 × height/2 × 64]. The activation map is then flatten and fed into 3 fully connected layers with an output size of 1024, 128, and 10 respectively. Between each layer, ReLU activtions are employed. The network is visualized in Figure Figure A.1.

The image classification model is trained with backpropagation to minimize the negative log likelihood (NLL) loss. Adam [258] is used as optimization algorithm with a mini-batch size of 32 a learning rate of $5 \cdot 10^{-4}$. As soon as the loss reaches a plateau, the learning rate is reduced to $1 \cdot 10^{-4}$.

[258]: Kingma et al. (2017)

After the model is trained, the first two fully connected layers within the network are pruned with Hebbian learning based methods. The

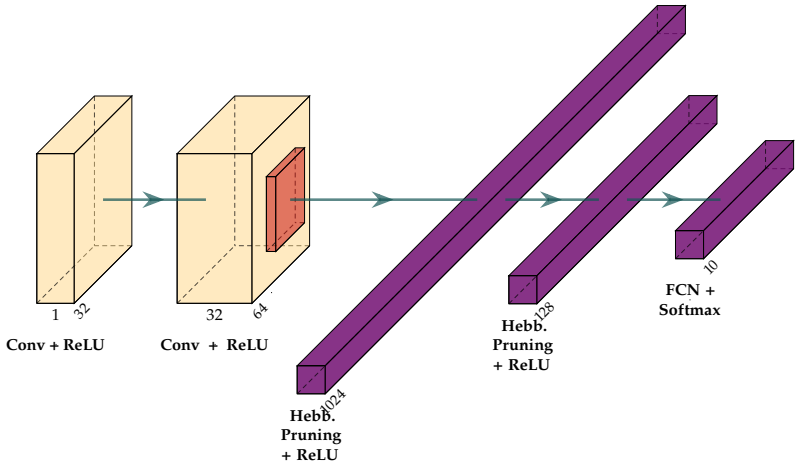

Figure A.1: The network architecture used to perform Hebbian-based weight pruning.

convolutional layers cannot be pruned because such layers employ a convolutional function that impedes calculating the correlation between an input and an output neuron. The last layer, on the other hand, cannot be pruned as it corresponds to the number of classes.

The *first* Hebbian based method proposed pushes the weights between neuron i and neuron j towards 0 if i and j have a low correlation within a mini-batch. First, a mini-batch is sampled. Afterwards, the correlation between i and j is calculated for each sample. if the correlation is below 0.02 for at least 20% of the samples (i.e. 7 samples or more for a mini-batch size of 32), the weight is updated as follows:

$$\Delta w_{ij} = -\eta_H \cdot w_{ij} \tag{A.2}$$

where η_H is the Hebbian learning rate set to $1 \cdot 10^{-5}$.

The *second* Hebbian based method proposed pushes some of the weights exactly to 0 while all other weights remain the same. During an epoch, the correlation between neurons i and j is measured for each weight w_{ij}. Afterwards, the weights with the lowest correlations are set to 0. How many weights are set to 0 is a hyper-parameter that has to be specified in advance.

For both methods, the model is pre-trained and pruned on the MNIST dataset [244] and evaluated on the MNIST as well as on the MNIST-C[1] dataset [262]. The *first* Hebbian based method can improve the results even after backpropagation as shown in Table Table **??**.

1: MNIST-C is a corrupted version of MNIST and well suited to measure model robustness

Table A.1: NLL loss of the model before and after the *first* Hebbian-based pruning method.

Dataset	NLL-Loss without Hebbian Updates	NLL-Loss with Hebbian Updates
MNIST	0.05314	0.03386
MNIST-C	0.6013	0.4766
Dataset	**Accuracy without Hebbian Updates**	**Accuracy with Hebbian Updates**
MNIST	98.94%	98.97%
MNIST-C	88.74%	89.94%

The NLL-loss on the MNIST test dataset decreases from 0.05314 to 0.03386, while the loss on MNIST-C test dataset decreases from 0.6013 to 0.4766 after applying the *first* Hebbian based method. However, in terms of accuracy the improvements are marginal. Furthermore, it has to be considered that this is only a preliminary experiment and this method certainly offers various potential for improvement. This experiments indicates that Hebbian in combination with back-propagation could lead to better or more robust representations because the Hebbian updates improved performance on both datasets.

The *second* Hebbian based method sets weights between neurons with a low correlation to 0. The network trained with back-propagation has an accuracy of 98.94% on MNIST. By setting up to 40% of the weights to 0, the accuracy remains above > 98%. Setting weights to 0 can make the network smaller and more efficient [263]. Furthermore, it leads to sparse representations that can improve robustness. Figure Figure A.2 visualizes the loss of the *second* Hebbian based method for different ratios of kept weights. Furthermore, the correlation based method is compared to randomly setting the same fraction of weights to 0.

[263]: Liang et al. (2021)

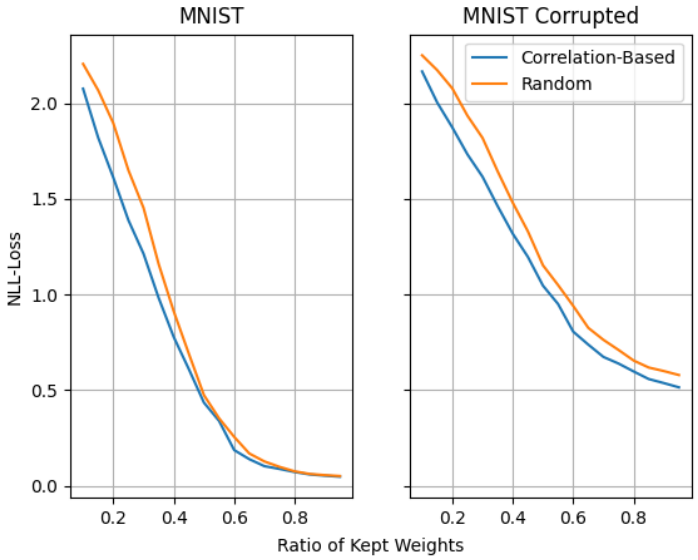

Figure A.2: The NLL loss of the *second* Hebbian based pruning method compared to removing random weights. The y axis shows the loss while the x axis shos the ratio of weights kept.

It can be observed that the performance is better when weights between neurons with low correlation are set to 0 than if random weights are set to 0.

Net-Fragments and Deep Learning

B

In this chapter, it is examined how representations of typical deep learning architectures look like. Especially net fragments are of interest, i.e. (groups of) neurons that represent certain features of objects. Net-fragments are discussed in Section 4.1.2. There, it is described that the interpretation of net-fragments in this thesis relies on two principles:

- The latent representations should be read out from multiple layers and not from a single layer
- To obtain meaningful activations, the activation maps should be sparse and diverse

The first principle is violated by most deep learning architectures as the latent representations are read out from a single layer and subsequently used for a downstream task. Since this first principle is violated by deep learning architectures by design, this chapter focuses mainly on the second principle. Specifically, it investigates whether specific neuron groups can be assigned to specific features (i.e. represent net-fragments). If this would be the case, specific features would be expected to trigger the activation of a group of neurons when they are present and that the same neurons would be inactive when this feature is not present. Thus, different neurons should be active for different objects. It is important to note that this mainly improves robustness and interpretability. Sparsity and diversity is not necessary, for example, to achieve high classification accuracy.

Deep Learning architectures usually consist of several layers with millions of neurons, leading to millions of activations [19–24]. Thus, the input data is processed in a complex way, which makes the analysis of activations difficult. To simplify the analysis of network activations, a novel straightforward classification dataset is proposed; The dataset consists of 10 images as shown in Figure B.1. Each image has a size of 9×9 pixels and depicts a number between 0 and 9. The images have only one channel and contain binary values (i.e. pixels are either set to 0 or 1). Small networks are sufficient to analyze these images and this dataset thus leads to less network activations what simplifies the search for net fragments.

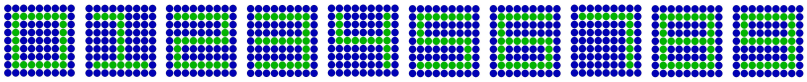

Figure B.1: A novel dataset created to investigate the properties of modern deep learning architectures. The dataset consists of images of the numbers $0-9$, each image has a size of 9×9 pixels. The blue dots represent pixels with the value 0, green dots represent pixels with the value 1.

Even in this small data set, there exist various features and consequently a multitude of network fragments. A very intuitive way for us humans to construct features on this data set is to interpret each line as a feature. For example, the dataset could be explained by 9 basic lines, that can be composed to 10 different digits. Thus, a low-level net fragment could

represent such a basic line and be composed to higher-level net fragments such as digits. Figure B.2 visualizes such an intuitive composition (basic lines on the left, digits on the right).

Thereby, each low-level fragment can be part of one or multiple digits. The composition can either take place across one or multiple layers. The visualization in Figure B.2 can be interpreted as a composition across one layer, i.e. the low-level fragments are combined to high-level fragments within one step.

The composition can also take place across multiple layers. An exemplary composition of net fragments of the number 9 across 4 subsequent layers is shown in Figure B.3. In this example, it is demonstrated that a fragment representing a digit can be composed of other fragments that also represent digits.

However, this explanatory example has to be understood as an intuitive way how humans would build net fragments. Actual net fragments do not necessarily have to be a composition of lines but can be a composition of multiple pixels without semantic. In fact, it can be expected that neural networks come up with their own features that look totally different than the example used above for explaining net fragments. Regardless of what these net fragments represent, the network should have certain characteristics; If a neuron or a group of neurons are part of a net fragment that represents a feature in the input, then these neuron(s) should be strongly active if the feature is present and not or only very weakly active if the feature is not present. This leads to a sparse activation map and net fragments should only be active for digits with specific characteristics. Thus, different neurons should be active depending on the digit.

Digits with common characteristics should have some overlapping network activities; For example, the 7 is contained in the 3, the 3 is contained in the 9 and the 9 is contained in the 8. Therefore, these digits should have some common network activations. The 1, on the other hand, has nothing in common with the 4, thus these digits should have no overlapping activations. In the following, networks are trained and their activations are examined for such properties.

Two types of models are trained on this dataset, namely a classification network (supervised) and autoencoder networks (unsupervised). The goal of the classification task is to predict the number (labels $0-9$) that is shown in the image (c.f. Section 6). This task is neither for humans nor deep learning architectures challenging. Usually, the last layer of an image classification architecture has exactly as many neurons as there are classes to predict. Each neuron corresponds to a class, and the neuron with the highest activity (i.e. the most active neuron) is eventually used to predict the image-level label. Such architectures thus have the design constraint that the last layer of neurons must represent distinct classes. Therefore, each neuron in the last layer can be interpreted as a representation of a class-object. Since the last layer of a classification network represents entire classes, such architectures seem to be well suited to investigate whether the preceding layers represent object features (i.e. net fragments) that are composed by the last layer.

A classification network relies on labels. An autoencoder, on the other hand, works unsupervised and reconstructs the input data from a lower

dimensional embedding space (c.f. section 3.4). Thus, the autoencoder generates representations that contain not only the class information but the entire image information. Consequently, the embedding activations of autoencoders are also examined in the embedding space. In addition, the autoencoder offers the possibility to constrain the embedding space so that net fragments are more strongly encouraged.

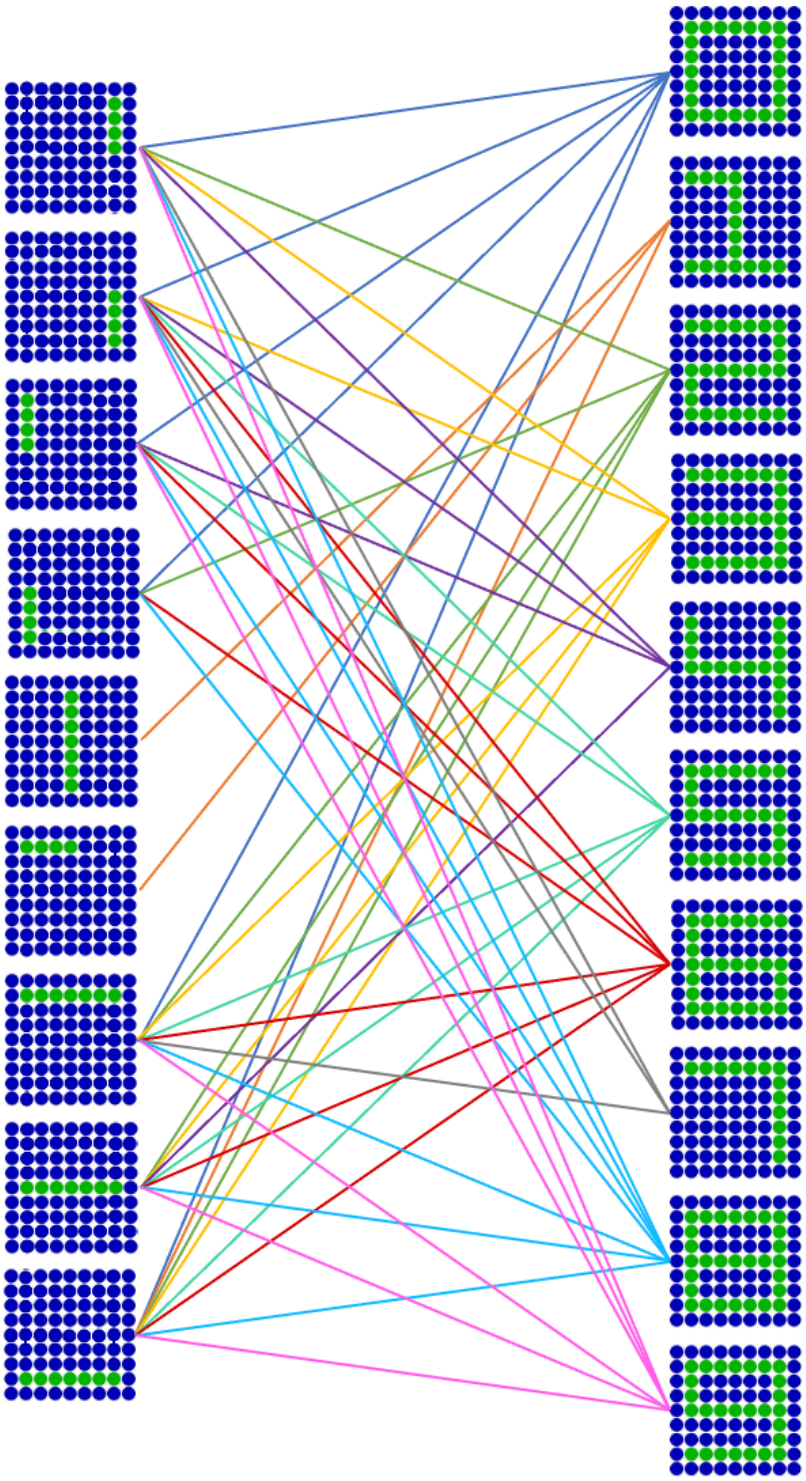

Figure B.2: A visualisation of all the lines (left) needed to compose the digits in the data set (right). The coloured lines in-between illustrate the relationship between the lines and the digits.

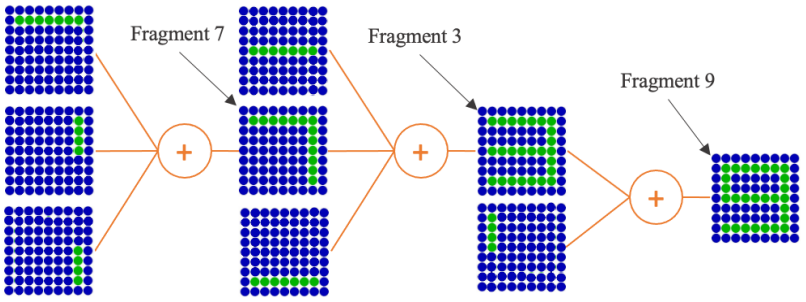

Figure B.3: A sample composition of the net fragment that could represent the digit 9.

B.1 Classification

B.1.1 Methods

To investigate the emergence of net fragments in classification networks, different architectures are trained. The model's parameters are optimised by minimising the cross-entropy loss with the Adam optimizer [10] so that the models learn to predict the corresponding class of the images shown in Figure B.1. The learning rate is $5 \cdot 10^{-4}$, the mini-batch size is 32 samples and the model is trained for a total of 10 epochs.

[10]: Kingma et al. (2015)

The models used have different feature extractors consisting of either convolutional (Conv.) layers (models no. 1-8, no. 11) or fully connected (FC) layers (models no. 10) as shown in Table **??**. Model no. 9 has no feature extractor as the input images are so simple that they can be considered as features by themself, model no. 10 uses a FC layer as feature extractor, and model no. 11 has a Conv. layer with two hand-crafted kernels to extract horizontal and vertical lines as feature extractor. After the feature extractor, all models have a similar "head" consisting of 2 fully connected layers. The feature extractor aims to extract certain features from the image, that are combined into higher-level net fragments in the first fully connected layer of the "head" and composed into predictions per class (i.e. net fragments corresponding to classes) in the last fully connected layer of the "head". The first FC layer of the "head" maps the input activations to 12 output features. The number of output features is determined empirically by training various architectures and examining the number of active neurons. It was found that there are always fewer than 12 neurons active and that this capacity is therefore sufficient. The last fully connected layer consists of 10 neurons since this corresponds to the number of classes to predict. The encoders of the models used are described in more detail in Table **??**. The "head" is identical for all models and consists of a sequence of the following layers; FC (out size=12) → ReLU → FC (out size=10) → Softmax.

Table B.1: A description of the different classification networks that are investigated for net-fragments.

No.	Encoder Description
1	Conv. Layer (kernel size=3 × 3, channels=2) → ReLU → head
2	Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → head
3	Conv. Layer (kernel size=5 × 5, channels=2) → ReLU → head
4	Conv. Layer (kernel size=5 × 5, channels=4) → ReLU → head
5	Conv. Layer (kernel size=3 × 3, channels=2) → ReLU → Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → head
6	Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → Conv. Layer (kernel size=3 × 3, channels=8) → ReLU → head
7	Conv. Layer (kernel size=3 × 3, channels=2) → ReLU → Max Pooling → Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → head
8	Conv. Layer (kernel size=3 × 3, channels=4) → ReLU → Max Pooling → Conv. Layer (kernel size=3 × 3, channels=8) → ReLU → head
9	head
10	FC (in size=9 ∗ 9, out size=12) → ReLU → head
11	Hand Crafted Conv. Layer for vertical & horizontal edge detection (kernel size=3 × 3, channels=2) → ReLU → head

After each epoch, the model's weights are stored as well as the activations of each layer. These vectors are visualized and investigated for net fragments.

Furthermore, the most relevant input features for a fully trained model are analysed. This is done by freezing the model's parameters so that they cannot change. Instead, an empty image is fed into the network and updated with backpropagation of error such that the probability for a given class is maximized. This leads to an input image that has the highest probability to be predicted by the model as a specific class (e.g. generate the image that has the highest probability to be predicted as digit 3 by the model).

B.1.2 Results

All the models learn to classify these digits perfectly within a few epochs. Interestingly, not only the accuracy reaches 100% but some models also achieve a cross-entropy loss of 0.0, meaning that they can find a global minimum. However, the goal is not to achieve high accuracy but to exhibit net fragments. It is not feasible to visualise all activations of all layers in this thesis. Therefore, only the activations of the second last FC layer (i.e. the first FC layer of the "head") after the ReLU function are shown. Since the last layer contains the net fragments that depict classes, this is the layer with the highest-level fragments. Furthermore, it is the only layer that has a global view on the input, i.e. can access all the features extracted by the encoder[1] (except for the encoder consisting of FC layers only). Some higher-level net fragments should be visible in this layer, which are subsequently composed in the last layer to net fragments corresponding to classes.

1: convolutional layers only consider a local neighbourhood by sliding a kernel over the input (c.f. Section 2.2.1)

Figure B.4 shows the activations of the first fully connected layer in the "head" of the different models. The FC layer has 12 neurons whose activation is indicated along the vertical axis (per model), and the horizontal axis depicts the activation of the same neuron for the classes 0-9. For example, the circle in the top left corner indicates the activation of the first neuron for class 0, the circle in the top right corner the activation of the first neuron for class 9, and the circle in the bottom left corner the activation of the last neuron for class 0. Blue circles show activations that are exactly 0, red circles are low activations > 0, and green circles represent strong activations[2].

2: activations < 0 do not exist because they are set to exactly 0 by the ReLU function

It can be seen in the visualization that none of the models utilises all 12 neurons and certain neurons are always inactive regardless of the class. Also, no obvious net fragments can be identified; Most neurons are always similarly strongly active regardless of the class. If net fragments are formed, however, the neurons of a fragment would have to be strongly active in the presence of certain features and weakly active otherwise. Such behaviour is not observable in these activations.

Furthermore, some digits are very similar and differ in only two pixels. Some examples are the digit pairs 5 and 6, 8 and 9, 0 and 8, or 3 and 9. However, when the activations of the corresponding classes are compared, it is obvious that almost always all activations change a little bit, and not one neuron is turned on or off depending on the presence of these two pixels.

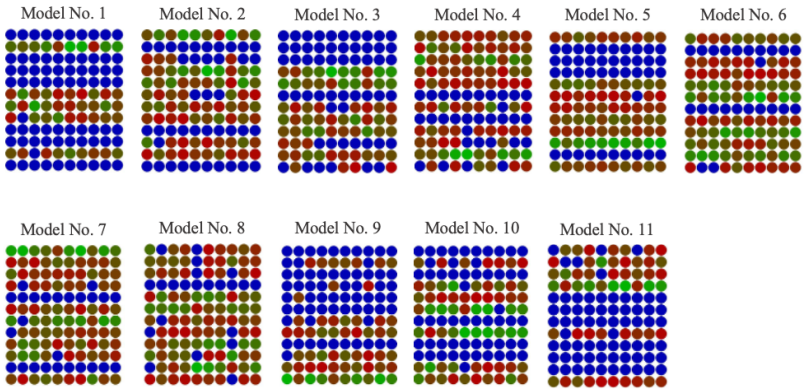

Figure B.4: The activations of the first fully connected layer in the "head" of the different networks. For each model, the activity of the 12 neurons is shown for each class (activations along the vertical axis, classes along the horizontal axis). Red means low activation, green means high activation, blue means activation is off (i.e. 0).

Figure B.5 visualizes the input that maximizes the probability for each class. It is obvious that all models focus on the wrong features and it can be assumed that these networks would not be robust to slight perturbations in the input. This also indicates that net fragments are not present in current deep learning models (or at least not to the desirable extent); if a class-level net fragment is composed of several lower-level net fragments, then some of these lower-level net fragments have to be active. Since these lower-level net fragments represent specific input features, corresponding pixel constellations should be visible in this visualisation. However, since these rather random-looking pixel combinations maximise the probability of predicting a specific class, this suggests that pixel combinations are not composed into hierarchically more complex net fragments.

Figure B.5: The inputs that maximize the probability that the different models predict a specific class. For example, the image in the top left corner maximizes the probability that model number 1 predicts that this is class 0 (digit 0).

B.2 Autoencoders

B.2.1 Methods

Similar to the experiments with classification networks, different autoencoders are also investigated. Since most architectures lead to similar results for the classification networks, the focus is less on testing many different architectures but rather on incorporating different constraints that might encourage net-fragments. Incorporating constraints into an autoencoder can be done easily by adding a loss function that regulate the activations in the bottleneck layer. For a classification network, on the other hand, it is not obvious how such constraints should be added. The two different autoencoder architectures used are shown in Table **??**.

Table B.2: A description of the different autoencoder networks that are investigated for net-fragments.

No.	Encoder Description
1	FC (in size=$9 * 9$ out size=12) $\rightarrow$ ReLU $\rightarrow$ FC (in size=12 out size=$9 * 9$)
2	FC (in size=$9 * 9$ out size=40) $\rightarrow$ ReLU $\rightarrow$ FC (in size=40 out size=$9 * 9$)

The autoencoder's parameters are optimised by minimising a reconstruction loss (i.e. a distance measurement between the input and the reconstructed output) with the Adam optimizer [10] so that the output looks similar to the input even tough the networks capacity is reduced by a bottleneck-layer in the middle. The learning rate is $5 \cdot 10^{-4}$, the mini-batch size is 32 samples and the model is trained for a total of 10 epochs.

[10]: Kingma et al. (2015)

The reconstruction loss L_{rec} used for the different experiments is either the L1 distance (mean absolute error) or the L2 distance (mean square

error). If $\boldsymbol{x}$ is the input data with N pixels and $\hat{\boldsymbol{x}}$ is the reconstructed data, then the reconstruction loss can be written as follows:

$$L_{rec}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} |x_n - \hat{x}_n|, & \text{if use L1 distance} \\ \frac{1}{N}\sum_{n=1}^{N} (x_n - \hat{x}_n)^2, & \text{if use L2 distance} \end{cases} \tag{B.1}$$

Optionally, a sparsity and a diversity loss were added to the reconstruction loss to encourage net-fragments. Since specific neurons of a net-fragment represent certain features, they should only be active if the feature is present. Good features only occur as part of a few objects and not in all of them: Thus, the neurons representing this feature should be active only sporadically and therefore the activations should be sparse and diverse. This combination of sparsity and diversity is also in line with the ideas of LeCun to obtain autonomous machine intelligence [260].

[260]: LeCun (2022)

For the sparsity loss L_s the kullback-leibler (KL) divergence is used [177]. The hidden representations $\boldsymbol{z}$ in the bottleneck layer with length m (i.e. m neurons in the bottleneck layer) are defined as $\boldsymbol{z} = z_1, ..., z_m$. Furthermore, the average activation probability of a neuron z_j over the input data $\boldsymbol{x}$ can be calculated as

[177]: Le et al. (2012)

$$\hat{\rho}_j = \frac{1}{N}\sum_{i}^{N} \frac{1}{1 + e^{-z_i^{(l)}}} \tag{B.2}$$

where $\frac{1}{1+e^{-z^{(i)}}}$ is the sigmoid function that squeezes the activation in the range between 0 and 1. Furthermore, $\rho = 0.05$ is sparsity parameter that can be interpreted as a target activation probability of each neuron. If the kullback-leibler divergence between all $\hat{\rho}_j$ and ρ is minimised, then an average activation probability per neuron in the hidden layer of ρ is enforced. The corresponding kullback-leibler divergence for each $\hat{\rho}_j$ is defined as:

$$KL(\rho||\hat{\rho}_j) = \rho \cdot \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \cdot \log \frac{1-\rho}{1-\hat{\rho}_j} \tag{B.3}$$

The sparsity loss L_s is the sum of the kullback-leibler divergence between all $\hat{\rho}_j$ and ρ and thus enforces that on average only 5% of the hidden representations $\boldsymbol{z}$ are active:

$$L_s(\rho, \hat{\rho}) = \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j) \tag{B.4}$$

Thus, this constraint ensures that the activations are sparse, which is consistent with the concept of net-fragments. Another property of net-fragments is that the activations should be diverse. Therefore, a new diversity loss L_d is proposed. This loss is based on the fact that the activations in the bottleneck layers $(\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(N)})$ should be as different as possible for all samples of a mini-batch $(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(N)})$. It is easy to see that the sum of the dot product between the activation of a sample $\boldsymbol{z}^{(i)}$ and all other samples $\boldsymbol{z}^{(j)}$, $i \neq j$ is small if the samples within a batch have a high diversity. Therefore, the diversity loss can be defined as:

$$L_d(\boldsymbol{z}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} \boldsymbol{z}^{(i)} \cdot \boldsymbol{z}^{(j)}, \text{ if } i \neq j \tag{B.5}$$

Minimising L_d results in the bottleneck layer activations not only being sparse but also diverse. The overall loss can be defined as the sum of these three loss components, with λ_s and λ_d beeing hyperparameters that control the influence of the diversity and sparsity loss.

$$L = L_{rec} + \lambda_s \cdot L_s + \lambda_d \cdot L_d \tag{B.6}$$

In the experiments, these hyperparameters were set to $lambda_s = 0.02$ and $lambda_d = 0.02$, respectively $lambda_s = 0$ and $lambda_d = 0$ if the sparsity and diversity loss should not be used.

Similar to the analysis of classification architectures, the model's activations of each layer are stored after every epoch. These vectors are visualized and investigated for net fragments.

B.2.2 Results

The activations of the autoencoders trained with reconstruction loss only and without diversity or sparsity constraint (i.e. $lambda_s = 0$ and $lambda_d = 0$) look very similar to those of the classification network; about half of the neurons are always active regardless of the object depicted in the image and the other half of neurons are always inactive. Thus, this version of the autoencoder has the same problem as the classification networks. Adding a sparsity constraint (i.e. $lambda_s = 0.02$ and $lambda_d = 0$) improves this issue slightly. The activations become sparse, but the same neurons are always active independent of the objects in the image. Only in a few cases can a set of neurons be identified that represent specific objects or features.

However, if all three loss components are used, the activations look much better. In this case, specific neuron combinations represent object-dependent features and are only active for specific objects. There are no more neurons that are constantly active. It is even possible to infer image labels based on a binary activity state (i.e. which neuron is active or inactive) and without knowing the strength of the activation, even though labels were not used during training. Thus, an autoencoder with additional sparsity and diversity constraint can represent net-fragments to some extent[3].

3: except that constraint 1 which says that net-fragments span several layers by design is violated (c.f. Introduction in Section **??**)

This can be observed in Figure B.6 that shows the activations of the "model 1". On the left are the activations without constraint, in the middle the activations with sparsity constraint and on the right the activations with sparsity and diversity constraint. For each of these three loss functions, the activations in the bottleneck layer are shown per class (activations along the vertical axis, classes along the horizontal axis). It is clearly visible how adding the sparsity constraint and the diversity constraint gives more meaning to the individual neurons. While without both constraints the same neurons are always active independent of the class, the activations are significantly more varied when both constraints

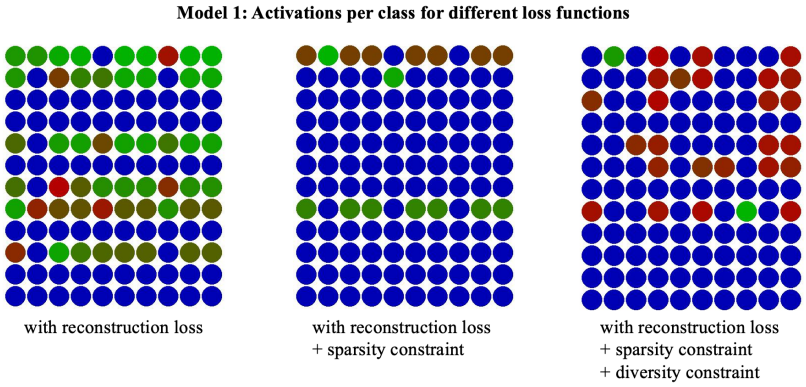

Figure B.6: The activations in the bottleneck layer of the smaller autoencoder "model 1" for different loss functions. For each loss function, the activity of the 12 neurons in the bottleneck layer is shown for each class (activations along the vertical axis, classes along the horizontal axis). Red means low activation, green means high activation, blue means activation is off (i.e. 0).

are used. This becomes even more obvious when the activations of the bigger autoencoder "model 2" are investigated (c.f. Figure B.7 (a)).

It is also examined what features the various neurons in the bottleneck layer represent. This is done by manually creating a bottleneck activation $\boldsymbol{z} = z_1, \ldots, z_m$ and feeding it through the decoder. To examine which feature the i-th neuron represents, $z_i = 1$ is set and $z_j = 0$, for $i \neq j$. Figure B.7 (b) shows which feature the 18-th neuron represents. Together with the 33-th neuron, this neuron represents the number 5 and together with the 36-th neuron the number 9. Thus, it is a feature (i.e. a net-fragment) that assembles with other features to represent an object.

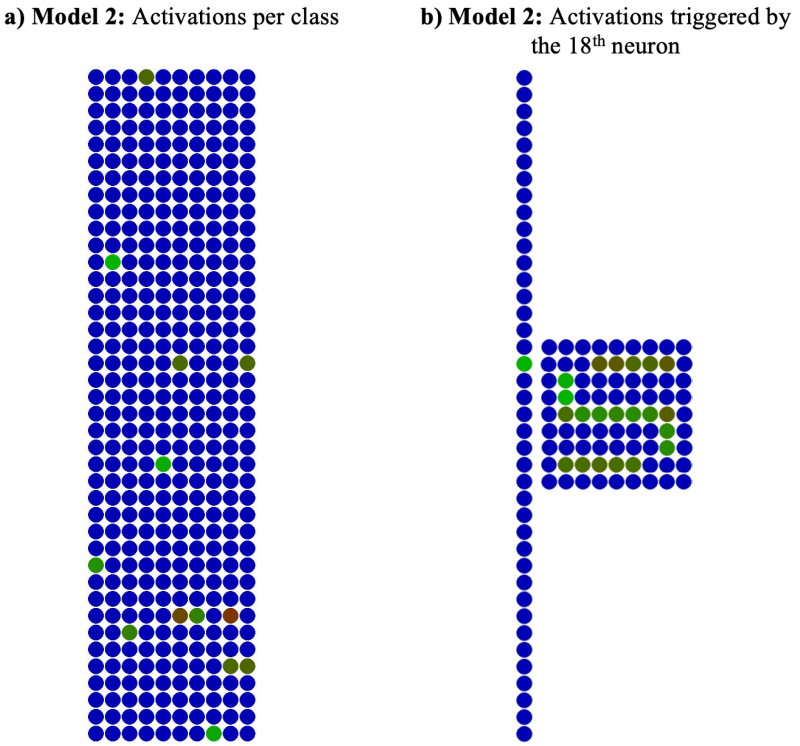

Figure B.7: On the lieft sinde (a), the activations in the bottleneck layer of the bigger autoencoder "model 2" are shown. The activity of the 40 neurons in the bottleneck layer is shown for each class (activations along the vertical axis, classes along the horizontal axis). On the right side (b), the output of the decoder is shown when the 18th neuron is set to 1 and all other neurons are set to 0. The 18-th neurons is a feature that is needed to generate the numbers 5 and 9.

B.3 Conclusion

Different architectures have been trained with different targets on a very simple and therefore easily interpretable data set. The network activations are tracked during training and analyzed for net-fragments. Typical vision architectures are sequential and build up a composition of features (i.e. net-fragments) over several layer. Usually, the embeddings of a pre-defined layer are used for down-stream tasks such as classification. Autoencoders typically use the bottleneck layer, a classification networks typically have a classification head (i.e. a FC layer with a Softmax activation) after the last model layer and thus extract these embeddings from the last model layer. Thus, embeddings are extracted from one layer. Net-fragments, on the other hand, span over multiple layers[4] and hence this principle is violated.

4: net-fragments can be thought of as the "path" of features through a network

Furthermore, net fragments are strongly active when the corresponding feature they represent is present in the input and are weakly active or not active at all when it is not present. Typical deep learning networks have neurons that are never active, while the active neurons usually remain active regardless of the class of the input data and only change their activity slightly. Thus, the information about different features is not distributed on different neurons but transported as activation strength of neurons through the network. Therefore, deep learning architectures do not comprise brain-like net fragments by default.

However, it was found that adding a sparsity and diversity constraint can alleviate this issue. Adding such constraints to the loss function encourages neurons to represent specific features that are typical for some of the objects.

Bibliography

Here is the list of references in citation order.

[1] Axios Media Inc. *Artificial intelligence pioneer says we need to start over*. 2017. URL: https://www.axios.com/2017/12/15/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524 (visited on 09/04/2022) (cited on page 1).

[2] Christoph von der Malsburg, Thilo Stadelmann, and Benjamin F. Grewe. 'A Theory of Natural Intelligence'. In: arXiv:2205.00002 (Apr. 2022). arXiv:2205.00002 [cs, q-bio] (cited on pages 2, 21, 22, 26, 30, 35, 37, 39).

[3] Wikipedia. *Neuron*. 2023. URL: https://en.wikipedia.org/wiki/Neuron (visited on 02/19/2023) (cited on page 5).

[4] Hiroshi Takagi. 'Roles of ion channels in EPSP integration at neuronal dendrites'. en. In: *Neuroscience Research* 37.3 (July 2000), pp. 167–171. DOI: 10.1016/S0168-0102(00)00120-6 (cited on page 6).

[5] J. S. Coombs, J. C. Eccles, and P. Fatt. 'The specific ionic conductances and the ionic movements across the motoneuronal membrane that produce the inhibitory post-synaptic potential'. en. In: *The Journal of Physiology* 130.2 (Nov. 1955), pp. 326–373. DOI: 10.1113/jphysiol.1955.sp005412 (cited on page 6).

[6] Moheb Costandi. *Neuroplasticity*. The MIT Press essential knowledge series. Cambridge, MA: The MIT Press, 2016 (cited on page 6).

[7] Warren S. McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. en. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/BF02478259 (cited on page 6).

[8] F. Rosenblatt. 'The perceptron: A probabilistic model for information storage and organization in the brain.' en. In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519 (cited on page 6).

[9] G. Cybenko. 'Approximation by superpositions of a sigmoidal function'. en. In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/BF02551274 (cited on page 7).

[10] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *CoRR* abs/1412.6980 (2015) (cited on pages 8, 65, 68).

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 'Learning representations by back-propagating errors'. en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0 (cited on page 8).

[12] Coursera Inc. *Deep Learning Specialization*. 2022. URL: https://www.coursera.org/specializations/deep-learning (visited on 08/19/2022) (cited on page 8).

[13] Coenraad Mouton, Johannes C. Myburgh, and Marelie H. Davel. 'Stride and Translation Invariance in CNNs'. In: vol. 1342. arXiv:2103.10097 [cs]. 2020, pp. 267–281. DOI: 10.1007/978-3-030-66151-9_17 (cited on page 9).

[14] Wei Zhang et al. 'Parallel distributed processing model with local space-invariant interconnections and its optical architecture'. en. In: *Applied Optics* 29.32 (Nov. 1990), p. 4790. DOI: 10.1364/AO.29.004790 (cited on page 9).

[15] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: arXiv:1502.03167 (Mar. 2015). arXiv:1502.03167 [cs] (cited on pages 9, 27).

[16] Y. Lecun et al. 'Gradient-based learning applied to document recognition'. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. DOI: 10.1109/5.726791 (cited on pages 9, 10, 22, 23, 26, 30, 46).

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012 (cited on pages 9, 10).

[18] Karen Simonyan and Andrew Zisserman. 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: arXiv:1409.1556 (Apr. 2015). arXiv:1409.1556 [cs] (cited on pages 9, 10).

[19] Christian Szegedy et al. 'Going Deeper with Convolutions'. In: arXiv:1409.4842 (Sept. 2014). arXiv:1409.4842 [cs] (cited on pages 9, 10, 61).

[20] Kaiming He et al. 'Deep Residual Learning for Image Recognition'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cited on pages 9, 10, 61).

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 'U-Net: Convolutional Networks for Biomedical Image Segmentation'. In: arXiv:1505.04597 (May 2015). arXiv:1505.04597 [cs] (cited on pages 9, 10, 61).

[22] Kaiming He et al. 'Mask R-CNN'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322 (cited on pages 9, 10, 61).

[23] Wei Liu et al. 'SSD: Single Shot MultiBox Detector'. en. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Vol. 9905. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2 (cited on pages 9, 10, 61).

[24] Joseph Redmon et al. 'You Only Look Once: Unified, Real-Time Object Detection'. In: arXiv:1506.02640 (May 2016). arXiv:1506.02640 [cs] (cited on pages 9, 10, 61).

[25] Venture Beat. *New deep learning model brings image segmentation to edge devices*. 2023. URL: https://venturebeat.com/ai/new-deep-learning-model-brings-image-segmentation-to-edge-devices/ (visited on 02/19/2023) (cited on page 10).

[26] Huikai Wu et al. 'FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation'. In: arXiv:1903.11816 (Mar. 2019). arXiv:1903.11816 [cs] (cited on page 10).

[27] Niclas Simmler et al. 'A Survey of Un-, Weakly-, and Semi-Supervised Learning Methods for Noisy, Missing and Partial Labels in Industrial Vision Applications'. In: *2021 8th Swiss Conference on Data Science (SDS)*. Lucerne, Switzerland: IEEE, June 2021, pp. 26–31. DOI: 10.1109/SDS51136.2021.00012 (cited on page 10).

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985 (cited on pages 10, 31).

[29] Gordon E. Moore. 'Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.' In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (Sept. 2006), pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860 (cited on page 10).

[30] Open AI. *AI and Compute*. 2018. URL: https://openai.com/blog/ai-and-compute/ (visited on 08/19/2022) (cited on page 11).

[31] Suhas Kumar. 'Fundamental Limits to Moore's Law'. In: arXiv:1511.05956 (Nov. 2015). arXiv:1511.05956 [cond-mat] (cited on page 11).

[32] Matthew Peters et al. 'Deep Contextualized Word Representations'. en. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202 (cited on page 11).

[33] Tom Brown et al. 'Language Models are Few-Shot Learners'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901 (cited on page 11).

[34] Lambda. *OpenAI's GPT-3 Language Model: A Technical Overview*. 2021. URL: https://lambdalabs.com/blog/demystifying-gpt-3/ (visited on 08/19/2022) (cited on page 11).

[35] Mohammad Shoeybi et al. 'Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism'. In: arXiv:1909.08053 (Mar. 2020). arXiv:1909.08053 [cs] (cited on page 11).

[36] Hao Wu et al. 'Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation'. In: arXiv:2004.09602 (Apr. 2020). arXiv:2004.09602 [cs, stat] (cited on page 11).

[37] Tejalal Choudhary et al. 'A comprehensive survey on model compression and acceleration'. en. In: *Artificial Intelligence Review* 53.7 (Oct. 2020), pp. 5113–5155. DOI: 10.1007/s10462-020-09816-7 (cited on page 11).

[38] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 'Distilling the Knowledge in a Neural Network'. In: arXiv:1503.02531 (Mar. 2015). arXiv:1503.02531 [cs, stat] (cited on page 11).

[39] Yu Zhang and Qiang Yang. 'A Survey on Multi-Task Learning'. In: arXiv:1707.08114 (Mar. 2021). arXiv:1707.08114 [cs] (cited on page 11).

[40] Doyen Sahoo et al. 'Online Deep Learning: Learning Deep Neural Networks on the Fly'. In: arXiv:1711.03705 (Nov. 2017). arXiv:1711.03705 [cs] (cited on page 11).

[41] German I. Parisi et al. 'Continual lifelong learning with neural networks: A review'. en. In: *Neural Networks* 113 (May 2019), pp. 54–71. DOI: 10.1016/j.neunet.2019.01.012 (cited on page 11).

[42] Spandan Madan et al. 'When and how convolutional neural networks generalize to out-of-distribution category–viewpoint combinations'. en. In: *Nature Machine Intelligence* 4.2 (Feb. 2022), pp. 146–153. DOI: 10.1038/s42256-021-00437-5 (cited on page 11).

[43] Gary Marcus. 'Deep Learning: A Critical Appraisal'. In: arXiv:1801.00631 (Jan. 2018). arXiv:1801.00631 [cs, stat] (cited on page 11).

[44] Hans Moravec. *Mind children: the future of robot and human intelligence*. eng. 4. print. Cambridge: Harvard Univ. Press, 1995 (cited on page 12).

[45] D. J. Felleman and D. C. Van Essen. 'Distributed Hierarchical Processing in the Primate Cerebral Cortex'. en. In: *Cerebral Cortex* 1.1 (Jan. 1991), pp. 1–47. DOI: 10.1093/cercor/1.1.1 (cited on page 13).

[46] Timothy P. Lillicrap et al. 'Random synaptic feedback weights support error backpropagation for deep learning'. en. In: *Nature Communications* 7.1 (Dec. 2016), p. 13276. DOI: 10.1038/ncomms13276 (cited on pages 12, 30).

[47] D. O. Hebb. *The organization of behavior; a neuropsychological theory.* The organization of behavior; a neuropsychological theory. Oxford, England: Wiley, 1949, pp. xix, 335 (cited on page 13).

[48] El Bienenstock, Ln Cooper, and Pw Munro. 'Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex'. en. In: *The Journal of Neuroscience* 2.1 (Jan. 1982), pp. 32–48. DOI: 10.1523/JNEUROSCI.02-01-00032.1982 (cited on page 14).

[49] Nathan Intrator and Leon N Cooper. 'Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions'. en. In: *Neural Networks* 5.1 (Jan. 1992), pp. 3–17. DOI: 10.1016/S0893-6080(05)80003-6 (cited on page 14).

[50] Erkki Oja. 'Simplified neuron model as a principal component analyzer'. en. In: *Journal of Mathematical Biology* 15.3 (Nov. 1982), pp. 267–273. DOI: 10.1007/BF00275687 (cited on page 14).

[51] Eero P Simoncelli and Bruno A Olshausen. 'Natural Image Statistics and Neural Representation'. en. In: *Annual Review of Neuroscience* 24.1 (Mar. 2001), pp. 1193–1216. DOI: 10.1146/annurev.neuro.24.1.1193 (cited on page 14).

[52] T. P. Vogels et al. 'Inhibitory Plasticity Balances Excitation and Inhibition in Sensory Pathways and Memory Networks'. en. In: *Science* 334.6062 (Dec. 2011), pp. 1569–1573. DOI: 10.1126/science.1211095 (cited on page 14).

[53] Prashant Joshi and Jochen Triesch. 'Rules for information maximization in spiking neurons using intrinsic plasticity'. In: *2009 International Joint Conference on Neural Networks*. 2009, pp. 1456–1461. DOI: 10.1109/IJCNN.2009.5178625 (cited on page 14).

[54] Michael Teichmann and Fred Hamker. 'Intrinsic plasticity: A simple mechanism to stabilize Hebbian learning in multilayer neural networks.' In: Mar. 2015 (cited on page 14).

[55] Stephen Grossberg. 'Nonlinear neural networks: Principles, mechanisms, and architectures'. en. In: *Neural Networks* 1.1 (Jan. 1988), pp. 17–61. DOI: 10.1016/0893-6080(88)90021-4 (cited on page 15).

[56] J J Hopfield. 'Neural networks and physical systems with emergent collective computational abilities.' en. In: *Proceedings of the National Academy of Sciences* 79.8 (Apr. 1982), pp. 2554–2558. DOI: 10.1073/pnas.79.8.2554 (cited on pages 15, 16).

[57] Evelyn Fix and J. L. Hodges. 'Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties'. In: *International Statistical Review / Revue Internationale de Statistique* 57.3 (Dec. 1989), p. 238. DOI: 10.2307/1403797 (cited on page 15).

[58] Jason Weston, Sumit Chopra, and Antoine Bordes. 'Memory Networks'. In: arXiv:1410.3916 (Nov. 2015). arXiv:1410.3916 [cs, stat] (cited on page 15).

[59] R. McEliece et al. 'The capacity of the Hopfield associative memory'. In: *IEEE Transactions on Information Theory* 33.4 (1987), pp. 461–482. DOI: 10.1109/TIT.1987.1057328 (cited on page 16).

[60] J. J. Hopfield, D. I. Feinstein, and R. G. Palmer. ''Unlearning' has a stabilizing effect in collective memories'. In: *Nature* 304.5922 (July 1983), pp. 158–159. DOI: 10.1038/304158a0 (cited on page 16).

[61] Dmitry Krotov and John J. Hopfield. 'Dense Associative Memory for Pattern Recognition'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 1180–1188 (cited on page 16).

[62] Mete Demircigil et al. 'On a Model of Associative Memory with Huge Storage Capacity'. en. In: *Journal of Statistical Physics* 168.2 (July 2017), pp. 288–299. DOI: 10.1007/s10955-017-1806-y (cited on page 16).

[63] Hubert Ramsauer et al. 'Hopfield Networks is All You Need'. In: arXiv:2008.02217 (Apr. 2021). arXiv:2008.02217 [cs, stat] (cited on page 16).

[64] Volodymyr Mnih et al. 'Recurrent Models of Visual Attention'. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2204–2212 (cited on page 16).

[65] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. en. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cited on page 16).

[66] Corinna Cortes and Vladimir Vapnik. 'Support-vector networks'. en. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. DOI: 10.1007/BF00994018 (cited on page 17).

[67] T. Cover and P. Hart. 'Nearest neighbor pattern classification'. In: *IEEE Transactions on Information Theory* 13.1 (Jan. 1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964 (cited on page 17).

[68] L. F. Abbott. 'Lapicque's introduction of the integrate-and-fire model neuron (1907)'. In: *Brain Research Bulletin* 50 (1999), pp. 303–304 (cited on page 17).

[69] E.M. Izhikevich. 'Simple model of spiking neurons'. en. In: *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440 (cited on pages 17, 26).

[70] Romain Brette and Wulfram Gerstner. 'Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity'. en. In: *Journal of Neurophysiology* 94.5 (Nov. 2005), pp. 3637–3642. DOI: 10.1152/jn.00686.2005 (cited on page 17).

[71] Hélène Paugam-Moisy. 'Spiking Neuron Networks A survey'. In: (2006) (cited on page 17).

[72] Guo-qiang Bi and Mu-ming Poo. 'Synaptic Modification by Correlated Activity: Hebb's Postulate Revisited'. en. In: *Annual Review of Neuroscience* 24.1 (Mar. 2001), pp. 139–166. DOI: 10.1146/annurev.neuro.24.1.139 (cited on page 17).

[73] Saeed Reza Kheradpisheh et al. 'STDP-based spiking deep convolutional neural networks for object recognition'. In: *Neural Networks* 99 (2018), pp. 56–67. DOI: https://doi.org/10.1016/j.neunet.2017.12.005 (cited on pages 17, 26).

[74] Herbert Jaeger. 'The" echo state" approach to analysing and training recurrent neural networks-with an erratum note''. In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148 (Jan. 2001) (cited on page 18).

[75] Wolfgang Maass, Thomas Natschläger, and Henry Markram. 'Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations'. en. In: *Neural Computation* 14.11 (Nov. 2002), pp. 2531–2560. DOI: 10.1162/089976602760407955 (cited on page 18).

[76] Zoran Konkoli. 'Reservoir Computing'. en. In: *Unconventional Computing*. Ed. by Andrew Adamatzky. New York, NY: Springer US, 2018, pp. 619–629. DOI: 10.1007/978-1-4939-6883-1_683 (cited on page 18).

[77] Gouhei Tanaka et al. 'Recent advances in physical reservoir computing: A review'. en. In: *Neural Networks* 115 (July 2019), pp. 100–123. DOI: 10.1016/j.neunet.2019.03.005 (cited on page 18).

[78] P. Erdös and A. Rényi. 'On Random Graphs I'. In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290 (cited on page 18).

[79] Mantas Lukoševičius. 'A Practical Guide to Applying Echo State Networks'. en. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 659–686. DOI: 10.1007/978-3-642-35289-8_36 (cited on page 18).

[80] John D. McPherson et al. 'A physical map of the human genome'. In: *Nature* 409.6822 (Feb. 2001), pp. 934–941. DOI: 10.1038/35057157 (cited on page 21).

[81] A.N. Kolmogorov. 'On tables of random numbers'. en. In: *Theoretical Computer Science* 207.2 (Nov. 1998), pp. 387–395. DOI: 10.1016/S0304-3975(98)00075-9 (cited on page 21).

[82] D. J. Willshaw and Christoph Von Der Malsburg. 'How patterned neural connections can be set up by self-organization'. en. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 194.1117 (Nov. 1976), pp. 431–445. DOI: 10.1098/rspb.1976.0087 (cited on page 21).

[83] D. J. Willshaw and Christoph Von Der Malsburg. 'A marker induction mechanism for the establishment of ordered neural mappings: its application to the retinotectal problem'. en. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 287.1021 (Nov. 1979), pp. 203–243. DOI: 10.1098/rstb.1979.0056 (cited on page 21).

[84] C. von der Malsburg and E Bienenstock. 'A Neural Network for the Retrieval of Superimposed Connection Patterns'. In: *Europhysics Letters (EPL)* 3.11 (June 1987), pp. 1243–1249. DOI: 10.1209/0295-5075/3/11/015 (cited on page 21).

[85] C. von der Malsburg. 'Concerning the Neuronal Code'. In: *Journal of Cognitive Science* 19.4 (Dec. 2018), pp. 511–550. DOI: 10.17791/JCS.2018.19.4.511 (cited on page 21).

[86] Walter J Freeman III and Christine A Skarda. 'Representations: Who needs them?' In: (1990) (cited on page 21).

[87] D. W. Arathorn. *Map-seeking circuits in visual cognition: a computational mechanism for biological and machine vision*. Stanford, Calif: Stanford University Press, 2002 (cited on page 22).

[88] Bruno A. Olshausen, Charles H. Anderson, and David C. Van Essen. 'A multiscale dynamic routing circuit for forming size- and position-invariant object representations'. In: *Journal of Computational Neuroscience* 2.1 (Mar. 1995), pp. 45–62. DOI: 10.1007/BF00962707 (cited on page 22).

[89] Tomas Fernandes and Christoph von der Malsburg. 'Self-Organization of Control Circuits for Invariant Fiber Projections'. en. In: *Neural Computation* 27.5 (May 2015), pp. 1005–1032. DOI: 10.1162/NECO_a_00725 (cited on page 22).

[90] Claude Lehmann. 'Leveraging Neuroscience for Deep Learning Based Object Recognition'. MA thesis. Zurich University of Applied Sciences, 2022 (cited on page 22).

[91] Marco Dorigo and Luca Maria Gambardella. 'Ant colony system: a cooperative learning approach to the traveling salesman problem'. In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66 (cited on page 23).

[92] Edvinas Byla and Wei Pang. 'DeepSwarm: Optimising Convolutional Neural Networks Using Swarm Intelligence'. en. In: *Advances in Computational Intelligence Systems*. Ed. by Zhaojie Ju et al. Vol. 1043. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, pp. 119–130. DOI: 10.1007/978-3-030-29933-0_10 (cited on page 23).

[93] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017 (cited on page 23).

[94] Alex Krizhevsky, Geoffrey Hinton, et al. 'Learning multiple layers of features from tiny images'. In: (2009) (cited on pages 23, 30).

[95] Stephen Wolfram. 'Cellular automata as models of complexity'. In: *Nature* 311.5985 (Oct. 1984), pp. 419–424. DOI: 10.1038/311419a0 (cited on page 23).

[96] Gérard Y. Vichniac. 'Simulating physics with cellular automata'. In: *Physica D: Nonlinear Phenomena* 10.1 (1984), pp. 96–116. DOI: https://doi.org/10.1016/0167-2789(84)90253-7 (cited on page 23).

[97] N. H. Wulff and John A. Hertz. 'Learning Cellular Automation Dynamics with Neural Networks'. In: *NIPS*. 1992 (cited on page 23).

[98] William Gilpin. 'Cellular automata as convolutional neural networks'. In: *Phys. Rev. E* 100 (3 Sept. 2019), p. 032402. DOI: 10.1103/PhysRevE.100.032402 (cited on page 23).

[99] Alexander Mordvintsev et al. 'Growing Neural Cellular Automata'. In: *Distill* (2020) (cited on page 23).

[100] Alexander Mordvintsev, Ettore Randazzo, and Craig Fouts. 'Growing Isotropic Neural Cellular Automata'. In: arXiv:2205.01681 (June 2022). arXiv:2205.01681 [cs, q-bio] (cited on page 23).

[101] Distill. *Growing Neural Cellular Automata*. 2022. URL: https://distill.pub/2020/growing-ca (visited on 09/05/2022) (cited on page 23).

[102] Rasmus Berg Palm et al. 'Variational Neural Cellular Automata'. In: arXiv:2201.12360 (Feb. 2022). arXiv:2201.12360 [cs] (cited on page 23).

[103] Diederik P. Kingma and Max Welling. 'Auto-Encoding Variational Bayes'. In: arXiv:1312.6114 (May 2014). arXiv:1312.6114 [cs, stat] (cited on pages 23, 31).

[104] Shyam Sudhakaran et al. 'Growing 3D Artefacts and Functional Machines with Neural Cellular Automata'. In: arXiv:2103.08737 (June 2021). arXiv:2103.08737 [cs] (cited on page 23).

[105] Kazuya Horibe, Kathryn Walker, and Sebastian Risi. 'Regenerating Soft Robots through Neural Cellular Automata'. In: arXiv:2102.02579 (Feb. 2021). arXiv:2102.02579 [cs, q-bio] (cited on page 23).

[106] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 'Learning Graph Cellular Automata'. In: arXiv:2110.14237 (Oct. 2021). arXiv:2110.14237 [cs] (cited on page 23).

[107] Jie Zhou et al. 'Graph Neural Networks: A Review of Methods and Applications'. In: arXiv:1812.08434 (Oct. 2021). arXiv:1812.08434 [cs, stat] (cited on page 23).

[108] Ettore Randazzo et al. 'Self-classifying MNIST Digits'. In: *Distill* (2020). https://distill.pub/2020/selforg/mnist. DOI: 10.23915/distill.00027.002 (cited on page 23).

[109] Alexandre Variengien et al. 'Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent'. In: arXiv:2106.15240 (July 2021). arXiv:2106.15240 [cs] (cited on page 24).

[110] Volodymyr Mnih et al. 'Playing Atari with Deep Reinforcement Learning'. In: arXiv:1312.5602 (Dec. 2013). arXiv:1312.5602 [cs] (cited on page 24).

[111] Elias Najarro and Sebastian Risi. 'Meta-Learning through Hebbian Plasticity in Random Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 20719–20731 (cited on pages 24, 57).

[112] Joachim Winther Pedersen and Sebastian Risi. 'Evolving and Merging Hebbian Learning Rules: Increasing Generalization by Decreasing the Number of Rules'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. arXiv:2104.07959 [cs]. June 2021, pp. 892–900. DOI: 10.1145/3449639.3459317 (cited on page 24).

[113] Louis Kirsch and Jürgen Schmidhuber. 'Meta Learning Backpropagation And Improving It'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021 (cited on pages 24, 33).

[114] Sebastian Risi. 'The Future of Artificial Intelligence is Self-Organizing and Self-Assembling'. In: *sebastianrisi.com* (2021) (cited on page 24).

[115] Teuvo Kohonen. 'Self-organized formation of topologically correct feature maps'. en. In: *Biological Cybernetics* 43.1 (1982), pp. 59–69. DOI: 10.1007/BF00337288 (cited on page 24).

[116] Teuvo Kohonen. *Self-Organization and Associative Memory*. eng. Third edition. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989 (cited on pages 24, 27).

[117] Bernd Fritzke. 'A Growing Neural Gas Network Learns Topologies'. In: *Advances in Neural Information Processing Systems*. Ed. by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press, 1994 (cited on page 25).

[118] Douglas L. Reilly, Leon N. Cooper, and Charles Elbaum. 'A neural model for category learning'. en. In: *Biological Cybernetics* 45.1 (Aug. 1982), pp. 35–41. DOI: 10.1007/BF00387211 (cited on page 25).

[119] Bernd Fritzke. 'Growing cell structures—A self-organizing network for unsupervised and supervised learning'. en. In: *Neural Networks* 7.9 (Jan. 1994), pp. 1441–1460. DOI: 10.1016/0893-6080(94)90091-4 (cited on page 25).

[120] Stephen Marsland, Jonathan Shapiro, and Ulrich Nehmzow. 'A self-organising network that grows when required'. en. In: *Neural Networks* 15.8–9 (Oct. 2002), pp. 1041–1058. DOI: 10.1016/S0893-6080(02)00078-3 (cited on page 25).

[121] Luiza Mici, German I. Parisi, and Stefan Wermter. 'A self-organizing neural network architecture for learning human-object interactions'. en. In: *Neurocomputing* 307 (Sept. 2018), pp. 14–24. DOI: 10.1016/j.neucom.2018.04.015 (cited on page 25).

[122] Mike Davies et al. 'Loihi: A Neuromorphic Manycore Processor with On-Chip Learning'. In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359 (cited on page 25).

[123] Timothée Masquelier and Simon J Thorpe. 'Unsupervised learning of visual features through spike timing dependent plasticity'. In: *PLoS computational biology* 3.2 (2007), e31 (cited on page 25).

[124] Qiang Yu et al. 'Rapid Feedforward Computation by Temporal Encoding and Learning With Spiking Neurons'. In: *IEEE Transactions on Neural Networks and Learning Systems* 24.10 (2013), pp. 1539–1552. DOI: 10.1109/TNNLS.2013.2245677 (cited on page 25).

[125] Saeed Reza Kheradpisheh et al. 'STDP-based spiking deep convolutional neural networks for object recognition'. In: *Neural Networks* 99 (2018), pp. 56–67 (cited on page 25).

[126] Milad Mozafari et al. 'Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks'. In: *Pattern recognition* 94 (2019), pp. 87–95 (cited on page 25).

[127] Peter U Diehl and Matthew Cook. 'Unsupervised learning of digit recognition using spike-timing-dependent plasticity'. In: *Frontiers in computational neuroscience* 9 (2015), p. 99 (cited on page 25).

[128] Yongqiang Cao, Yang Chen, and Deepak Khosla. 'Spiking deep convolutional neural networks for energy-efficient object recognition'. In: *International Journal of Computer Vision* 113.1 (2015), pp. 54–66 (cited on page 25).

[129] Amirhossein Tavanaei and Anthony S Maida. 'Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning'. In: *arXiv preprint arXiv:1611.03000* (2016) (cited on page 25).

[130] Peter U Diehl et al. 'Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing'. In: *2015 International joint conference on neural networks (IJCNN)*. ieee. 2015, pp. 1–8 (cited on page 25).

[131] Friedemann Zenke and Surya Ganguli. 'Superspike: Supervised learning in multilayer spiking neural networks'. In: *Neural computation* 30.6 (2018), pp. 1514–1541 (cited on page 25).

[132] Paul Ferré, Franck Mamalet, and Simon J Thorpe. 'Unsupervised feature learning with winner-takes-all based stdp'. In: *Frontiers in computational neuroscience* 12 (2018), p. 24 (cited on page 25).

[133] Milad Mozafari et al. 'First-spike-based visual categorization using reward-modulated STDP'. In: *IEEE transactions on neural networks and learning systems* 29.12 (2018), pp. 6178–6190 (cited on page 25).

[134] Guruprasad Raghavan, Cong Lin, and Matt Thomson. 'Self-organization of multi-layer spiking neural networks'. In: arXiv:2006.06902 (June 2020). arXiv:2006.06902 [cs, q-bio] (cited on page 26).

[135] Guruprasad Raghavan and Matt Thomson. 'Neural networks grown and self-organized by noise'. In: *NeurIPS*. 2019 (cited on page 26).

[136] Ruthvik Vaila, John Chiasson, and Vishal Saxena. 'Deep Convolutional Spiking Neural Networks for Image Classification'. In: arXiv:1903.12272 (Sept. 2019). arXiv:1903.12272 [cs] (cited on page 26).

[137] Francis Crick. 'The recent excitement about neural networks'. en. In: *Nature* 337.6203 (Jan. 1989), pp. 129–132. DOI: 10.1038/337129a0 (cited on pages 27, 35).

[138] Stephen Grossberg. 'Competitive Learning: From Interactive Activation to Adaptive Resonance'. en. In: *Cognitive Science* 11.1 (Jan. 1987), pp. 23–63. DOI: 10.1111/j.1551-6708.1987.tb00862.x (cited on pages 27, 35).

[139] Jingzhao Zhang et al. 'Why gradient clipping accelerates training: A theoretical justification for adaptivity'. In: arXiv:1905.11881 (Feb. 2020). arXiv:1905.11881 [cs, math] (cited on page 27).

[140] Chen-Yu Lee et al. 'Deeply-Supervised Nets'. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, May 2015, pp. 562–570 (cited on page 27).

[141] Bernard Widrow et al. 'Nature's Learning Rule'. en. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 1–30. DOI: 10.1016/B978-0-12-815480-9.00001-3 (cited on page 27).

[142] Kamil A. Grajski and Michael M. Merzenich. 'Hebb-Type Dynamics is Sufficient to Account for the Inverse Magnification Rule in Cortical Somatotopy'. en. In: *Neural Computation* 2.1 (Mar. 1990), pp. 71–84. DOI: 10.1162/neco.1990.2.1.71 (cited on page 27).

[143] P. Read Montague, Joseph A. Gally, and Gerald M. Edelman. 'Spatial Signaling in the Development and Function of Neural Connections'. en. In: *Cerebral Cortex* 1.3 (1991), pp. 199–220. DOI: 10.1093/cercor/1.3.199 (cited on page 27).

[144] Bartlett W. Mel. 'NMDA-Based Pattern Discrimination in a Modeled Cortical Neuron'. en. In: *Neural Computation* 4.4 (July 1992), pp. 502–517. DOI: 10.1162/neco.1992.4.4.502 (cited on page 27).

[145] Russell W. Anderson. 'Biased Random-Walk Learning: A Neurobiological Correlate to Trial-and-Error'. en. In: *Neural Networks and Pattern Recognition*. Elsevier, 1998, pp. 221–244. DOI: 10.1016/B978-012526420-4/50008-2 (cited on page 27).

[146] David E. Rumelhart and James L. McClelland. 'Learning Internal Representations by Error Propagation'. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362 (cited on page 27).

[147] Stephen Grossberg and Nestor A. Schmajuk. 'Neural dynamics of adaptive timing and temporal discrimination during associative learning'. en. In: *Neural Networks* 2.2 (Jan. 1989), pp. 79–102. DOI: 10.1016/0893-6080(89)90026-9 (cited on page 27).

[148] Shiyu Duan, Shujian Yu, and Jose C. Principe. 'Modularizing Deep Learning via Pairwise Learning With Kernels'. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.4 (Apr. 2022), pp. 1441–1451. DOI: 10.1109/TNNLS.2020.3042346 (cited on page 27).

[149] Shiyu Duan et al. 'On Kernel Method–Based Connectionist Models and Supervised Deep Learning Without Backpropagation'. en. In: *Neural Computation* 32.1 (Jan. 2020), pp. 97–135. DOI: 10.1162/neco_a_01250 (cited on page 27).

[150] Yulin Wang et al. 'Revisiting Locally Supervised Learning: an Alternative to End-to-end Training'. In: arXiv:2101.10832 (Jan. 2021). arXiv:2101.10832 [cs, stat] (cited on page 27).

[151] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 'Greedy layerwise learning can scale to imagenet'. In: *International conference on machine learning*. PMLR. 2019, pp. 583–593 (cited on pages 27, 28).

[152] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. 'Deep Supervised Learning Using Local Errors'. In: *Frontiers in Neuroscience* 12 (Aug. 2018), p. 608. DOI: 10.3389/fnins.2018.00608 (cited on page 27).

[153] Enrique S. Marquez, Jonathon S. Hare, and Mahesan Niranjan. 'Deep Cascade Learning'. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (Nov. 2018), pp. 5475–5485. DOI: 10.1109/TNNLS.2018.2805098 (cited on page 27).

[154] Arild Nokland and Lars Hiller Eidnes. 'Training Neural Networks with Local Error Signals'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 2019, pp. 4839–4850 (cited on page 27).

[155] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 'Decoupled Greedy Learning of CNNs'. In: *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020 (cited on page 28).

[156] Geoffrey Hinton. *The Forward-Forward Algorithm: Some Preliminary Investigations*. 2022. URL: https://www.cs.toronto.edu/~hinton/FFA13.pdf (visited on 12/17/2022) (cited on pages 28, 49).

[157] Geoffrey Hinton. 'How to represent part-whole hierarchies in a neural network'. In: arXiv:2102.12627 (Feb. 2021). arXiv:2102.12627 [cs] (cited on page 28).

[158] Yoshua Bengio. 'How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation'. In: arXiv:1407.7906 (Sept. 2014). arXiv:1407.7906 [cs] (cited on page 29).

[159] Dong-Hyun Lee et al. 'Difference Target Propagation'. en. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Annalisa Appice et al. Vol. 9284. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 498–515. DOI: 10.1007/978-3-319-23528-8_31 (cited on pages 29, 30).

[160] Alexander Meulemans et al. 'A Theoretical Framework for Target Propagation'. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020 (cited on page 29).

[161] Max Jaderberg et al. 'Decoupled Neural Interfaces using Synthetic Gradients'. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 1627–1635 (cited on page 30).

[162] Wojciech Marian Czarnecki et al. 'Understanding Synthetic Gradients and Decoupled Neural Interfaces'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 904–912 (cited on page 30).

[163] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. 'Learning to solve the credit assignment problem'. In: arXiv:1906.00889 (Apr. 2020). arXiv:1906.00889 [cs, q-bio] (cited on page 30).

[164] Arild Nokland. 'Direct Feedback Alignment Provides Learning in Deep Neural Networks'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 1045–1053 (cited on page 30).

[165] Will Xiao et al. 'Biologically-plausible learning algorithms can scale to large datasets.' In: *International Conference on Learning Representations, (ICLR 2019)*. 2019 (cited on page 30).

[166] David Balduzzi, Hastagiri Vanchinathan, and Joachim M. Buhmann. 'Kickback Cuts Backprop's Red-Tape: Biologically Plausible Credit Assignment in Neural Networks'. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 485–491 (cited on page 30).

[167] Qianli Liao, Joel Z. Leibo, and Tomaso Poggio. 'How Important is Weight Symmetry in Backpropagation?' In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 1837–1844 (cited on page 30).

[168] Sergey Bartunov et al. 'Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures'. In: arXiv:1807.04587 (Nov. 2018). arXiv:1807.04587 [cs, stat] (cited on pages 30, 36).

[169] Miguel Carreira-Perpinan and Weiran Wang. 'Distributed optimization of deeply nested systems'. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Ed. by Samuel Kaski and Jukka Corander. Vol. 33. Proceedings of Machine Learning Research. Reykjavik, Iceland: PMLR, Apr. 2014, pp. 10–19 (cited on page 30).

[170] Gavin Taylor et al. 'Training Neural Networks without Gradients: A Scalable ADMM Approach'. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 2722–2731 (cited on page 30).

[171] Ziming Zhang and Matthew Brand. 'Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1719–1728 (cited on page 30).

[172] Tim Tsz-Kit Lau et al. 'A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training'. In: arXiv:1803.09082 (Mar. 2018). arXiv:1803.09082 [cs, math, stat] (cited on page 30).

[173] Shiyu Duan and Jose C. Principe. 'Training Deep Architectures Without End-to-End Backpropagation: A Survey on the Provably Optimal Methods'. In: arXiv:2101.03419 (Aug. 2022). arXiv:2101.03419 [cs, stat] (cited on page 30).

[174] Jia Deng et al. 'ImageNet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Miami, FL: IEEE, June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cited on page 30).

[175] Pierre Baldi. 'Autoencoders, Unsupervised Learning, and Deep Architectures'. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon et al. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, July 2012, pp. 37–49 (cited on page 31).

[176] Marc'Aurelio Ranzato et al. 'Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition'. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. Minneapolis, MN, USA: IEEE, June 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383157 (cited on page 31).

[177] Quoc V. Le et al. 'Building High-Level Features Using Large Scale Unsupervised Learning'. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 507–514 (cited on pages 31, 42, 69).

[178] Pascal Vincent et al. 'Extracting and Composing Robust Features with Denoising Autoencoders'. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294 (cited on page 31).

[179] Salah Rifai et al. 'Contractive Auto-Encoders: Explicit Invariance during Feature Extraction'. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 833–840 (cited on page 31).

[180] Omar Elharrouss et al. 'Image Inpainting: A Review'. en. In: *Neural Processing Letters* 51.2 (Apr. 2020), pp. 2007–2028. DOI: 10.1007/s11063-019-10163-0 (cited on page 32).

[181] Deepak Pathak et al. 'Context Encoders: Feature Learning by Inpainting'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 2536–2544. DOI: 10.1109/CVPR.2016.278 (cited on page 32).

[182] Kaiming He et al. 'Masked autoencoders are scalable vision learners'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009 (cited on page 32).

[183] Yuge Shi et al. 'Adversarial masking for self-supervised learning'. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20026–20040 (cited on page 32).

[184] Nikos Komodakis and Spyros Gidaris. 'Unsupervised representation learning by predicting image rotations'. In: *International Conference on Learning Representations (ICLR)*. 2018 (cited on page 32).

[185] Xiaohua Zhai et al. 'S4L: Self-Supervised Semi-Supervised Learning'. In: arXiv:1905.03670 (July 2019). arXiv:1905.03670 [cs] (cited on page 32).

[186] Yanbei Chen et al. 'Semi-Supervised and Unsupervised Deep Visual Learning: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022) (cited on page 32).

[187] Ting Chen et al. 'A simple framework for contrastive learning of visual representations'. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607 (cited on pages 32, 41).

[188] Kaiming He et al. 'Momentum contrast for unsupervised visual representation learning'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738 (cited on page 32).

[189] Mathilde Caron et al. 'Unsupervised learning of visual features by contrasting cluster assignments'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9912–9924 (cited on pages 32, 41).

[190] Prannay Khosla et al. 'Supervised contrastive learning'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18661–18673 (cited on page 32).

[191] Timothy M Hospedales et al. 'Meta-Learning in Neural Networks: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/TPAMI.2021.3079209 (cited on pages 32, 33).

[192] Sebastian Thrun and Lorien Pratt. 'Learning to Learn: Introduction and Overview'. en. In: *Learning to Learn*. Ed. by Sebastian Thrun and Lorien Pratt. Boston, MA: Springer US, 1998, pp. 3–17. DOI: 10.1007/978-1-4615-5529-2_1 (cited on page 32).

[193] Allan M. Schrier. 'Learning how to learn: The significance and current status of learning set formation'. en. In: *Primates* 25.1 (Jan. 1984), pp. 95–102. DOI: 10.1007/BF02382299 (cited on page 32).

[194] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 'Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1126–1135 (cited on pages 33, 34).

[195] Chelsea Finn, Kelvin Xu, and Sergey Levine. 'Probabilistic Model-Agnostic Meta-Learning'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montreal, Canada: Curran Associates Inc., 2018, pp. 9537–9548 (cited on page 33).

[196] Yoonho Lee and Seungjin Choi. 'Gradient-based meta-learning with learned layerwise metric and subspace'. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2927–2936 (cited on page 33).

[197] Siyuan Qiao et al. 'Few-shot image recognition by predicting parameters from activations'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7229–7238 (cited on page 33).

[198] Andrei A Rusu et al. 'Meta-learning with latent embedding optimization'. In: *arXiv preprint arXiv:1807.05960* (2018) (cited on page 33).

[199] TM Heskes. 'Empirical Bayes for learning to learn'. In: (2000) (cited on page 33).

[200] James Requeima et al. 'Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes'. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cited on page 33).

[201] David Ha, Andrew Dai, and Quoc V. Le. 'HyperNetworks'. In: arXiv:1609.09106 (Dec. 2016). arXiv:1609.09106 [cs] (cited on page 33).

[202] Jake Snell, Kevin Swersky, and Richard Zemel. 'Prototypical Networks for Few-Shot Learning'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4080–4090 (cited on pages 33, 34).

[203] Wei-Yu Chen et al. 'A Closer Look at Few-shot Classification'. In: arXiv:1904.04232 (Jan. 2020). arXiv:1904.04232 [cs] (cited on page 33).

[204] Sachin Ravi and Hugo Larochelle. 'Optimization as a Model for Few-Shot Learning'. In: (2017) (cited on pages 33, 34).

[205] Ke Li and Jitendra Malik. 'Learning to Optimize'. In: arXiv:1606.01885 (June 2016). arXiv:1606.01885 [cs, math, stat] (cited on page 33).

[206] Zhenguo Li et al. 'Meta-SGD: Learning to Learn Quickly for Few-Shot Learning'. In: arXiv:1707.09835 (Sept. 2017). arXiv:1707.09835 [cs] (cited on page 33).

[207] Eunbyung Park and Junier B. Oliva. 'Meta-Curvature'. In: arXiv:1902.03356 (Jan. 2020). arXiv:1902.03356 [cs, stat] (cited on page 33).

[208] Rein Houthooft et al. 'Evolved Policy Gradients'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cited on page 33).

[209] Flood Sung et al. 'Learning to Learn: Meta-Critic Networks for Sample Efficient Learning'. In: arXiv:1706.09529 (June 2017). arXiv:1706.09529 [cs] (cited on page 33).

[210] Giulia Denevi et al. 'Learning To Learn Around A Common Mean'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cited on page 33).

[211] Giulia Denevi et al. 'Online-Within-Online Meta-Learning'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cited on page 33).

[212] Santiago Gonzalez and Risto Miikkulainen. 'Improved training speed, accuracy, and data utilization through loss function optimization'. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2020, pp. 1–8 (cited on page 33).

[213] Yiying Li et al. 'Feature-Critic Networks for Heterogeneous Domain Generalization'. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3915–3924 (cited on pages 33, 34).

[214] Antreas Antoniou and Amos J Storkey. 'Learning to Learn By Self-Critique'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cited on page 33).

[215] Rinu Boney and Alexander Ilin. 'Semi-Supervised Few-Shot Learning with MAML'. In: *International Conference on Learning Representations*. 2018 (cited on page 33).

[216] Barret Zoph and Quoc Le. 'Neural Architecture Search with Reinforcement Learning'. In: *International Conference on Learning Representations*. 2017 (cited on page 33).

[217] Justin Bayer et al. 'Evolving Memory Cell Structures for Sequence Learning'. In: *Artificial Neural Networks – ICANN 2009*. Ed. by Cesare Alippi et al. Vol. 5769. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 755–764. DOI: 10.1007/978-3-642-04277-5_76 (cited on page 33).

[218] Esteban Real et al. 'Regularized Evolution for Image Classifier Architecture Search'. In: AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33014780 (cited on page 33).

[219] Luca Franceschi et al. 'Bilevel Programming for Hyperparameter Optimization and Meta-Learning'. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 1568–1577 (cited on pages 33, 34).

[220] Paul Micaelli and Amos Storkey. 'Gradient-based Hyperparameter Optimization Over Long Horizons'. In: arXiv:2007.07869 (Sept. 2021). arXiv:2007.07869 [cs, stat] (cited on pages 33, 34).

[221] Luca Franceschi et al. 'Forward and Reverse Gradient-Based Hyperparameter Optimization'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1165–1173 (cited on pages 33, 34).

[222] Ekin D. Cubuk et al. 'AutoAugment: Learning Augmentation Strategies From Data'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019 (cited on pages 33, 34).

[223] Yonggang Li et al. 'Differentiable Automatic Data Augmentation'. en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Vol. 12367. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 580–595. DOI: 10.1007/978-3-030-58542-6_35 (cited on page 33).

[224] Cheng Zhang et al. 'Active Mini-Batch Sampling Using Repulsive Point Processes'. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33015741 (cited on page 33).

[225] Yang Fan et al. 'Learning to Teach'. In: *International Conference on Learning Representations*. 2018 (cited on page 33).

[226] Tongzhou Wang et al. 'Dataset Distillation'. In: arXiv:1811.10959 (Feb. 2020). arXiv:1811.10959 [cs, stat] (cited on page 33).

[227] Jonathan Lorraine, Paul Vicol, and David Duvenaud. 'Optimizing Millions of Hyperparameters by Implicit Differentiation'. In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, Aug. 2020, pp. 1540–1552 (cited on pages 33, 34).

[228] OpenAI: Marcin Andrychowicz et al. 'Learning dexterous in-hand manipulation'. en. In: *The International Journal of Robotics Research* 39.1 (Jan. 2020), pp. 3–20. DOI: 10.1177/0278364919887447 (cited on page 34).

[229] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. 'Learning To Simulate'. In: *International Conference on Learning Representations*. 2019 (cited on page 34).

[230] Quan Vuong et al. 'How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?' In: arXiv:1903.11774 (Mar. 2019). arXiv:1903.11774 [cs, stat] (cited on page 34).

[231] Chen Huang et al. 'Addressing the loss-metric mismatch with adaptive loss alignment'. In: *International conference on machine learning*. PMLR. 2019, pp. 2891–2900 (cited on page 34).

[232] Yan Duan et al. 'RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning'. In: arXiv:1611.02779 (Nov. 2016). arXiv:1611.02779 [cs, stat] (cited on page 34).

[233] Jurgen Schmidhuber. 'Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook'. Diploma Thesis. Technische Universitat Munchen, Germany, May 1987 (cited on page 34).

[234] Kenneth O. Stanley et al. 'Designing neural networks through neuroevolution'. en. In: *Nature Machine Intelligence* 1.1 (Jan. 2019), pp. 24–35. DOI: 10.1038/s42256-018-0006-z (cited on page 34).

[235] Tim Salimans et al. 'Evolution Strategies as a Scalable Alternative to Reinforcement Learning'. In: arXiv:1703.03864 (Sept. 2017). arXiv:1703.03864 [cs, stat] (cited on pages 34, 57).

[236] Olga Wichrowska et al. 'Learned Optimizers That Scale and Generalize'. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3751–3760 (cited on page 34).

[237] Antreas Antoniou, Harrison Edwards, and Amos Storkey. 'How to train your MAML'. In: *International Conference on Learning Representations*. 2019 (cited on page 34).

[238] JA Scott Kelso. *Dynamic patterns: The self-organization of brain and behavior*. MIT press, 1995 (cited on page 35).

[239] Birgitta Dresp. 'Seven Properties of Self-Organization in the Human Brain'. In: *Big Data Cogn. Comput.* 4 (2020), p. 10 (cited on page 35).

[240] Timothy P. Lillicrap et al. 'Random feedback weights support learning in deep neural networks'. In: arXiv:1411.0247 (Nov. 2014). arXiv:1411.0247 [cs, q-bio] (cited on page 35).

[241] Randall C. O'Reilly. 'Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm'. en. In: *Neural Computation* 8.5 (July 1996), pp. 895–938. DOI: 10.1162/neco.1996.8.5.895 (cited on page 35).

[242] Yann Le Cun. 'Learning Process in an Asymmetric Threshold Network'. en. In: *Disordered Systems and Biological Organization*. Ed. by E. Bienenstock, F. Fogelman Soulié, and G. Weisbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 233–240. DOI: 10.1007/978-3-642-82657-3_24 (cited on page 35).

[243] Jia Deng et al. 'Imagenet: A large-scale hierarchical image database'. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cited on page 36).

[244] Yann LeCun and Corinna Cortes. *THE MNIST DATABASE of handwritten digits*. 1996. URL: http://yann.lecun.com/exdb/mnist/ (visited on 09/05/2022) (cited on pages 36, 58).

[245] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 'CIFAR-10 (Canadian Institute for Advanced Research)'. In: () (cited on page 36).

[246] Javier R. Movellan. 'Contrastive Hebbian Learning in the Continuous Hopfield Model'. en. In: *Connectionist Models*. Elsevier, 1991, pp. 10–17. DOI: 10.1016/B978-1-4832-1448-1.50007-X (cited on page 36).

[247] Yu-An Chung et al. 'An Unsupervised Autoregressive Model for Speech Representation Learning'. In: arXiv:1904.03240 (June 2019). arXiv:1904.03240 [cs, eess] (cited on page 37).

[248] Nathalie L. Rochefort et al. 'Sparsification of neuronal activity in the visual cortex at eye-opening'. en. In: *Proceedings of the National Academy of Sciences* 106.35 (Sept. 2009), pp. 15049–15054. DOI: 10.1073/pnas.0907660106 (cited on page 39).

[249] Christos Louizos, Max Welling, and Diederik P. Kingma. 'Learning Sparse Neural Networks through L_0 Regularization'. In: arXiv:1712.01312 (June 2018). arXiv:1712.01312 [cs, stat] (cited on page 39).

[250] Torsten Hoefler et al. 'Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks'. In: arXiv:2102.00554 (Jan. 2021). arXiv:2102.00554 [cs] (cited on page 39).

[251] Konstantinos P. Panousis, Sotirios Chatzis, and Sergios Theodoridis. 'Stochastic Local Winner-Takes-All Networks Enable Profound Adversarial Robustness'. In: arXiv:2112.02671 (Dec. 2021). arXiv:2112.02671 [cs, stat] (cited on page 39).

[252] Charles D Gilbert, Joseph A Hirsch, and Torsten N Wiesel. 'Lateral interactions in visual cortex'. In: *Cold Spring Harbor symposia on quantitative biology*. Vol. 55. Cold Spring Harbor Laboratory Press. 1990, pp. 663–677 (cited on page 39).

[253] Ting Chen et al. 'Big self-supervised models are strong semi-supervised learners'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22243–22255 (cited on page 41).

[254] Georg B. Keller, Tobias Bonhoeffer, and Mark Hübener. 'Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse'. en. In: *Neuron* 74.5 (June 2012), pp. 809–815. DOI: 10.1016/j.neuron.2012.03.040 (cited on pages 41, 53).

[255] Jan Storck, Sepp Hochreiter, Jürgen Schmidhuber, et al. 'Reinforcement driven information acquisition in non-deterministic environments'. In: *Proceedings of the international conference on artificial neural networks, Paris*. Vol. 2. 1995, pp. 159–164 (cited on page 41).

[256] Magdalena Klapper-Rybicka, Nicol N. Schraudolph, and Jürgen Schmidhuber. 'Unsupervised Learning in LSTM Recurrent Neural Networks'. In: *Artificial Neural Networks — ICANN 2001*. Ed. by Georg Dorffner, Horst Bischof, and Kurt Hornik. Vol. 2130. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 684–691. DOI: 10.1007/3-540-44668-0_95 (cited on page 41).

[257] Vassileios Balntas et al. 'Learning local feature descriptors with triplets and shallow convolutional neural networks'. en. In: *Procedings of the British Machine Vision Conference 2016*. York, UK: British Machine Vision Association, 2016, pp. 119.1–119.11. DOI: 10.5244/C.30.119 (cited on page 44).

[258] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980 (cited on pages 45, 57).

[259] Hamza Keurti et al. 'Homomorphism Autoencoder – Learning Group Structured Representations from Observed Transitions'. In: arXiv:2207.12067 (July 2022). arXiv:2207.12067 [cs, math, stat] (cited on page 53).

[260] Yann LeCun. *A Path Towards Autonomous Machine Intelligence*. 2022. URL: https://openreview.net/forum?id=BZ5a1r-kVsf (visited on 09/21/2022) (cited on pages 53, 69).

[261] Yael Niv et al. 'Evolution of Reinforcement Learning in Uncertain Environments: Emergence of Risk-Aversion and Matching'. In: *Advances in Artificial Life*. Ed. by Jozef Kelemen and Petr Sosík. Vol. 2159. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 252–261. DOI: 10.1007/3-540-44811-X_27 (cited on page 57).

[262] Norman Mu and Justin Gilmer. 'MNIST-C: A Robustness Benchmark for Computer Vision'. In: arXiv:1906.02337 (June 2019). arXiv:1906.02337 [cs] (cited on page 58).

[263] Tailin Liang et al. 'Pruning and Quantization for Deep Neural Network Acceleration: A Survey'. In: arXiv:2101.09671 (June 2021). arXiv:2101.09671 [cs] (cited on page 59).