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

December 8, 2022

Zurich University of Applied Sciences

Dedicated to the dedicated.

– Pascal Sager

"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

This thesis is the last hurdle before I will hold the title “Master of Science”. To me, science means the systematic analysis of the real or virtual world through observations and experiments as well as the further development of existing technology. I am lucky enough to be able to apply the knowledge and methodologies I learned during my studies to research projects at the Centre for Artificial Intelligence (CAI) of the Zurich University of Applied Sciences (ZHAW). I was mentored and supported during my studies by Prof. Dr. Thilo Stadelmann. 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 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 are able to follow my reasoning.

... TODO ...

Contents

List of Figures

List of Tables

Introduction 1

Mankind has always tried to simplify its life through technological progress. Thousands of years before Christ the wheel was invented, later the pulley, the printing press, and the steam locomotive. At the beginning of the 19th century, the first generators were built to produce electricity, and in 1835, the first light bulb was invented by James Bowman Lindsay[1]. In the course of time more and more complex electrical circuits were produced and finally in 1941 the first digital calculating machine, the computer was developed. This calculating machine is able to perform various arithmetic operations on the basis of commands. Through the development of transistors, more and more complex computers could be produced, which are able to execute various tasks. The definition of these tasks are captured in software. 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 tools. 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 is clearly defined and can be described precisely. However, there exists tasks which are almost impossible to program. For example, writing a script that detects cats in images is almost impossible because we cannot describe how a cat looks like[2].

1: and not, as so often falsely claimed, by Thomas Alva Edision

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

Therefore, computer 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 are able to learn and adapt without following explicit instructions such as program code. Instead, they use statistical models to analyse data, to find patterns in the data and to draw inferences out of 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, namely Deep Learning, has made waves in the last decade. Breakthroughs are being made with this technology are made almost monthly and allows us to execute more and more tasks.

However, such DL system are usually good at one ore a few closely related tasks. In fact, they have they have some crucial flaws by definition (c.f. Section Section 2.2) 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 back propagation (c.f. Section Section 2.1) has shaped the field. Back propagation is *the* core learning algorithm of DL systems and was developed in 1986. More than

30 years later, in 2017, Hinton says that he is "deeply suspicious" about back propagation 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)

Mankind is often inspired by nature when it comes to developing novel systems. The development of artificial intelligence (AI) and Deep Learning has been strongly influenced by, according to our perception, one of (if not the) most intelligent systems on planet Earth, the human brain. The brain is studied by the scientific field of neuroscience[3]. From neuroscience or related fields come various insights on how the human nervous system and especially the brain works. Much of this knowledge has been gained through observations and experiments on humans or other living creatures. Often these findings are only what can be measured from the outside, the real core, the mechanism that causes intelligence, remains unknown.

3: there exist many additional (sub-)fields studying the brain such as cognitive science, cognitive psychology, neurology, and neuropsychology

While Deep Learning is clearly inspired by the insights of Neuroscience, the two fields have little in common in their present form. The implementation of neuroscientific findings has emerged as a subfield in its own right, called Neurocomputing (c.f. Section Section 2.4). In general, neurocomputing is closer related to the functionality of the human brain than Deep Learning. Neurocomputing has produced many promising algorithms that can overcome some weaknesses faced by deep learning systems. Although neurocomputing has also achieved significant breakthroughs, most systems used in everyday life are still based on the principle of Deep Learning.

TODO: deep learning or Deep Learning -> make uniform

1.0.1 Motivation

One of the main challenges of today's AI systems is the processing of visual information. Visual perception is fundamentally the task of finding a suitable interpretation of a given scene. A scene typically consists of objects that interact with each other. Identifying these objects and their relationship among each other is implemented by algorithms as a non-deterministic search problem, i.e. algorithms try to identify the object within a scene. State-of-the-art algorithms for image processing are often based on Deep Learning. Despite the fact that Deep Learning has achieved incredible performance on a variety of tasks such as object recognition it is still questionable whether the current methodology is enough to achieve real (or at least more advanced) intelligence (c.f. Section 2.2). Algorithms from the field of neurocomputing (c.f. Section 2.4), which are more closely oriented to the findings from the field of neuroscience, can be considered as a complement to Deep Learning. However, even tough algorithms from the field of neurocomputing have interesting properties, they usually do not perform as good as Deep Learning according to the evaluation metrics used and often work on specific or small data only.

Thus, various algorithms exist to solve the non-deterministic problem of finding an appropriate interpretation of a visual scene, but they

have significant shortcomings. Interestingly, the problem of generating a scene[4] from a clear description is deterministic. Computer graphics deals with the problem to generate images with the aid of computers. Nowadays, such systems are able to render photorealistic images based on descriptions. For example, video games are essentially programs that generate scenes consisting of multiple objects such as people, weapons, buildings, or vehicles based on program code. For each of these objects, skeletons are usually pre-defined and during rendering transformed (i.e. their size, rotation, distortion and positioning are adjusted) as well as properly illuminated (raytracing). This allows to generate an infinite number of scenes from a predefined set of skeletons and transformations. Essential in this process seems the separation between objects and object transformation. Therefore, incorporating these insights into current algorithms for the interpretation of visual scenes seems promising and could lead to significant improvements. More precisely, identifying objects and their transformations separately in preception systems in order to recognize objects independently of their transformations could improve visual scene interpretation.

4: the opposite task than finding an interpretation of a scene

It is known that large parts of the human brain are self-organizing [2]. Recently, renowned scientists [3] put forward the hypothesis that the key mechanism of natural intelligent systems such as the human brain to interpret visual scenes is 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. Dresp [4] 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.

[2]: Kelso (1995)
[3]: Malsburg et al. (2022)
[4]: Dresp (2020)

However, it is not obvious how these insights from neuroscience can be integrated into a Deep Learning framework. One interpretation of self-organization may be that instead of optimizing the entire network by statistical learning for a single task (i.e. using backpropagation), local optimization based on local input data takes place. The suitability of backpropagation for explaining how the brain learns was questioned soon after it was published [5, 6]. Many alternative and biologically more plausible algorithms have been proposed in recent years such as the feedback alignment (FA) algorithm [7], contrastive Hebbian learning [8], generalized recirculation [9], as well as target propagation (TP) and its variants [10–12]. However, Bartunov et al. [13] have shown that these algorithms do not scale to large vision datasets such as ImageNet [14] and only work for smaller datasets such as MNIST [15] and CIFAR-10 [16]. The biologically most plausible learning algorithm is Hebbian learning (c.f. Section Section 2.4.1). However, even tough I obtain some promising results in preliminary results with Hebbian learning (c.f. Appendix Section **??**), this algorithm doesn't seem to be well suited to learn good

[5]: Crick (1989)
[6]: Grossberg (1987)
[7]: Lillicrap et al. (2014)
[8]: Mazzoni et al. (1991)
[9]: O'Reilly (1996)
[10]: LeCun (1986)
[11]: Hinton et al. (2007)
[12]: Lee et al. (2015)
[13]: Bartunov et al. (2018)

image representation if a network is trained from scratch.

Another interpretation of self-organization within neural network is that the networks adapts its structure during training. According to von der Malsburg et al. [3], neurons form net-fragments (a.k.a. sub-networks) that represent objects with different levels of abstraction within an image (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. Thus, self-organization could be the algorithm to form net-fragments that represent objects independent of their transformation.

Such a mechanism to form networks that can capture scenes in the form of net-fragment has not been successfully implemented in deep learning nor neurocomputing systems. This thesis aims to develop a learning paradigm that explicitly foster creation of net-fragments that represent objects independent of their transformation. ... TODO ...

1.0.2 Terminology

This work is at the intersection of computer science and neuroscience. Deep Learning, which is the principle used in this thesis, is strongly inspired by the findings of neuroscience. Therefore, many terms and concepts overlap and are therefore defined in this chapter.

A biological neuron (i.e. found in the human body and animals) 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 [17] or inhibitory [18], 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 [19].

[18]: Takagi (2006) (1955)

[19]: 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 (c.f. Section Section 2.1). Several variants for artificial neurons have been proposed in the literature. These variants are described in the following chapter 2. Similar to biological neurons,

multiple artificial neurons are usually connected to an artificial neural network.

In this thesis, the two fields are declared as well as possible. If it is not clear from the context what we are talking about, we will refer to biological or artificial neurons and biological or artificial neural networks. This thesis describes the field of interest as clearly as possible. If it is not clear from the context to which field the referred entity belongs to, it is clarified by stating it (e.g. biological or artificial neuron). However, some concepts are identical in both fields. For example, there are excitatory or inhibitory signals or neural plasticity in biological and artificial neurons. Consequently, these terms are not distinguished.

In addition to these well-defined terms, there are various biological principles observed in the field of Neuroscience that are either not completely understood or it is unclear how they can be incorporated into artificial networks. The concepts relevant for this thesis are discussed in the following. However, due to the aforementioned challenges, no definition of these principles is given but rather my own interpretation in the context of this work.

A central concept of this work are sub-networks. Typically, in current DL architectures for computer vision, an encoder is used to compute object representations in a latent space. Thus, the output of a single layer can be understood as an object representation[5]. According to von der Malsburg et al. [3], biological sub-networks span multiple layers. This concept is interpreted in such a way that the latent representations should not be extracted from the last layer of an encoder but from several layers. While the first layers of an image encoder contain more local information, the last layers contain very global information. It is known from speech representation learning that different layers capture different speech information [20]. For example, the first layers contain more information to distinguish speakers, while representations in later layers provide more phonetic content. However, such information seems to be poorly exploited in computer vision architectures. One could argue that skip connections in segmentation networks [21] exploit such information. While this may be true to some extent, information from previous layers is still squeezed into a feature vector and can be poorly exploited by downstream tasks. TODO: Umsetzung Sub-Networks beschreiben

5: this is also true for image classification: Usually, a classification head consisting of fully connected layers and subsequent softmax activation is used after an image encoder

[20]: Chung et al. (2019)

[21]: Ronneberger et al. (2015)

In addition to forward connections, lateral connections are also located in visual cortex [22]. Thus, the biological neurons are not only connected to the neurons in the subsequent layer but also within the same layer. TODO: Je nach Umsetzung erklären, wie lateral connections in ANN interpretiert werden (+ Zeichnung einfügen)

[22]: Gilbert et al. (1990)

Another principle that seems important in biological neural networks is sparsity. Over time, the number of active neurons in the visual cortex decreases. 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 [23]. Thus, a sparsification of neuronal activations takes place through visual experience. In the field of Deep Learning, sparisty 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 [**Nun_Dryden_Peste_2021**, 24]. From a biological point

[23]: Rochefort et al. (2009)

of view, this process of first creating a large network and then shrinking it is obviously not plausible[6]. Sparse activations, on the other hand, can increase robustness [25]. Intuitively, sparse activations enforce that only the most relevant information is passed to the subsequent layer. Furthermore, sparse activations can help to obtain sub-networks. Objects are represented in sub-networks (i.e. in multiple layers). If all neurons in all layers are always active, then object representations cannot be extracted from sub-networks, since it is unclear which activation pattern represents which object. If, on the other hand, only a fraction of the neurons are active, then certain neuron combinations can infer objects. TODO: Umsetzung Sprase Activations beschreiben

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

A biological intelligent organism has an embodiment and can interact with the world. At the same time, the visual system perceives continuous input all the time (except when sleeping or blinking). As a result, the visual signal changes only minimally from one perceived frame to the next over a long course of time. 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 [26–28]. 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. TODO: Umsetzung beschreiben

[26]: Chen et al. (2020)
[27]: Chen et al. (2020)
[28]: Caron et al. (2020)

1.0.3 Contribution

TODO: Einzelnes Kapitel in dem beschrieben wird, was implementiert wurde, resp. wie dies hilft

1.0.4 Organization of Thesis

The remainder of the thesis is organized as follows: In chapter 2 we present the fundamentals necessary to understand this thesis. We provide an overview about how deep learning works and what the most common research areas in neurocomputing are. Chapter 3 presents the work related to this thesis.

TODO....

Fundamentals 2

TODO: Input Thilo -> mache den Bezug von diesem Teil auf die schlussendliche Arbeit viel deutlicher -> was wird wie gebraucht?

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. A sub-branch of Machine Learning is Deep Learning (DL). DL algorithms are able to learn hidden patterns within data to make predictions. They benefit from the accelerated computing power and big data made available in the last decade. Deep Learning is considered state-of-the-art for many learning tasks especially for high-dimensional data. Typical high-dimensional data are texts, audio recordings, 2D as well as 3D images, and videos. Deep Learning models use artificial neural networks to learn the mapping function between given input and output data. In the following, the fundamentals of Deep Learning is explained. Only those aspects that are relevant for the understanding of the rest of the thesis are discussed.

2.1 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 [29]. 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.

[29]: 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 [30] 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.

[30]: 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 (2.3) and (3.1) 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

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

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 [31] 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

[31] 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 [32]

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

The gradients of the weights $\boldsymbol{w}^{[l]}$ and biases $b^{[l]}$ can efficiently be calculated with an algorithm called backpropagation [33], which is just a smart implementation of the chain rule[4].

[33]: 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 [34]

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.

2.1.1 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 [36]. 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 [35]

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 [37], 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 [38], AlexNet [39], VGGNet [40], GoogLeNet [41], ResNet [42], U-Net [21], Mask R-CNN [43], SSD [44], and YOLO [45].

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

[37] Ioffe et al. (2015)
[38] LeCun et al. (1989)
[39] Krizhevsky et al. (2012)
[40] Simonyan and Zisserman (2014)
[41] Szegedy et al. (2015)
[42] He et al. (2016)
[43] He et al. (2017)
[44] Liu et al. (2016)
[45] 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. 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.

TODO: Bild einfügen mit diesen Tasks und darauf verweisen.

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) [38–42], object detection requires the coordinates of the bounding box [43–45], and semantic segmentation requires the labels on pixel-level [21, 46] (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 [47].

[47]: Simmler et al. (2021)

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 3.3.

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

2.2 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 [48] 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 [49]. 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 [50]. 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.

[50]: Kumar (2015)

Even the growth in the last five years is astonishing. While the state-of-the-art language model from 2018 [51] had around 94M parameters, the state-of-the-art in 2020 [52] already had 175B parameters. Training such a model on a single V100 GPU would take about 355 years and cost about 4.6M dollars [53]. A recent language model from Microsoft and Nvidia [54] 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 [55], model pruning [56], or model distillation [57] it is questionable if making models bigger is the best way to develop intelligent systems.

[51]: Peters et al. (2018)
[52]: Brown et al. (2020)
[54]: 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 [58] 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 [59] and lifelong learning [60] 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 [61].

9: Generalization refers to the ability of the model to adapt properly to previously unseen data from the same distribution
[61]: 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 [62] 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!!!
[62]: 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

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

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 [63] 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".

[63]: Moravec (1995)

2.3 Biological Learning

The human brain comprises many interconnected areas processing everything in parallel. For example, Figure Figure 2.1 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

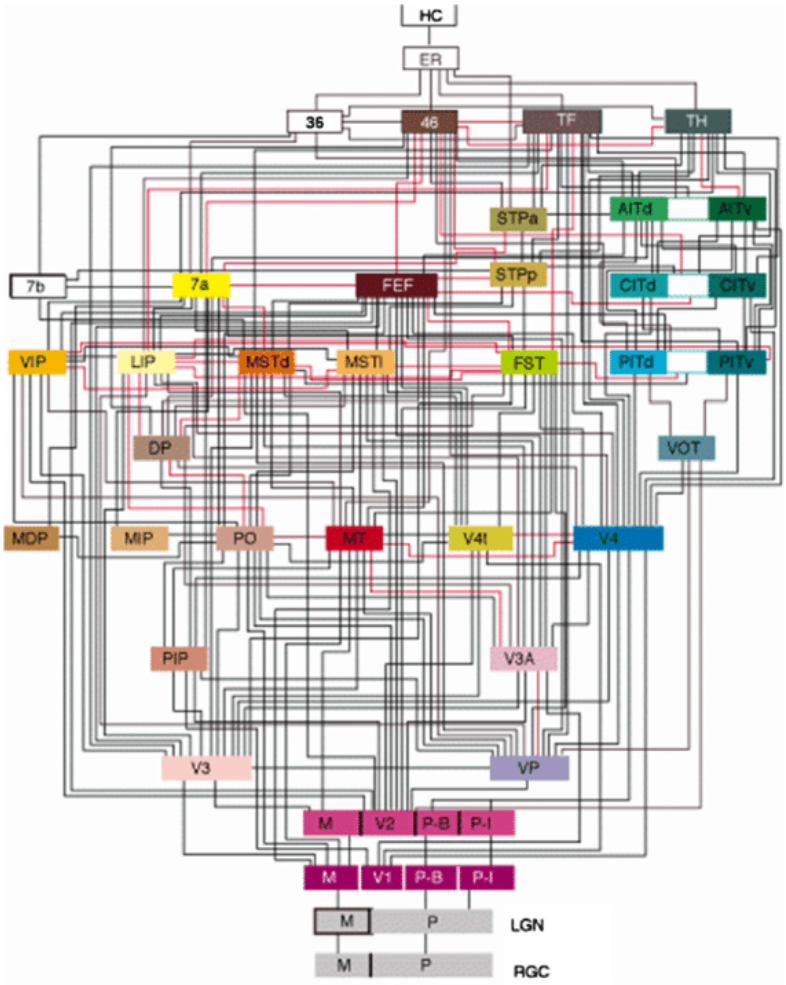

Figure 2.1: The organization of the visual system in the cerebral cortex. The image is from Felleman et al. [64].

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. [65] 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.

[65]: 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 (2.6)), 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.

TODO: Add reference to reservoir computing

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 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.

2.4 Neurocomputing

2.4.1 Hebbian Learning

[66]: Hebb (1949)

Donald Hebb [66] 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".

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. [67] and extended by Intrator and Cooper [68]. 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).

[68]: Intrator et al. (1992)

Around the same time, Oja [69] improved the learning rule of Equation (2.9) with an normalization term:

[69]: 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 [70]. 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 [71].

[70]: Simoncelli et al. (2001)

[71]: Vogels et al. (2011)

There exists many further improvements for Hebbian learning which are not summarized in this thesis. For example, Joshi and Triesch [72] as well as Teichmann and Hamker [73] adapt the activation function of the neurons to enforce a certain activity distribution and to stabilize Hebbian learning even in multilayer neural networks.

[72]: Joshi et al. (2009)

[73]: Teichmann et al. (2015)

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 [74] which leads to a combination of supervised and unsupervised Hebbian learning.

[74]: Grossberg (1988)

2.4.2 Hopfield Networks

Hopfield networks [75] 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 [76]. 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 [77] 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.

[75]: Hopfield (1982)

[76]: Fix et al. (1989)

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

[77]: 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 (2.14), 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.

McElice [78] 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 2.4.1) 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]:

[78]: McEliece et al. (1987)

16: for the derivation of this equation refer to [75]

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

where $\boldsymbol{I}$ is the identity matrix. Later, Hopfield et al. [79] extended the binary Hopfield network so that it can either learn pattern during an awake cycle or forget patterns during a sleep cycle.

[79]: Hopfield et al. (1983)

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 [80] 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 [81] and thus modern Hopfield networks have an exponential capacity of $C \approx 2^{\frac{N}{2}}$.

[80]: Krotov et al. (2016)

[81]: Demircigil et al. (2017)

The second limiting factor of binary Hopfield networks is that only binary patterns can be stored. Ramsauer et al. [82] 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, LSTM layers, support vector machines (SVM), and k-NN.

[82]: Ramsauer et al. (2021)

TODO: References for LSTM, attention, SVM, ...???

2.4.3 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 [83]. 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 [84] or the adaptive exponential IF [85]. 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

[83]: Abbott (1999)

[84]: Izhikevich (2003)

[85]: 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 [86].

[86]: Paugam-Moisy (2006)

The synaptic plasticity can be modeled with Hebbian learning (c.f. Section 2.4.1). The spike-timing dependent (STDP) plasticity rule [87] 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.

[87]: Bi et al. (2001)

For a long time, SNN only worked for very shallow networks. In 2018, Kheradpisheh et al. [88] 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 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

[88]: Kheradpisheh et al. (2018)

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

last convolutional layer are flatten and a Support Vector Machine is used to classify the features.

2.4.4 Reservoir Computing

As described in Section 2.3, 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) [89] and Liquid State Machines (LSM) [90]. 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 [91]. However, not every system is suited as reservoir. A good reservoir system distributes different inputs into different regions of the computation space [91].

[89]: Jaeger (2001)
[90]: Maass et al. (2002)
[91]: Konkoli (2018)

A ESN is a set of sparsely connected recurrent neurons as visualized in Figure 2.2.

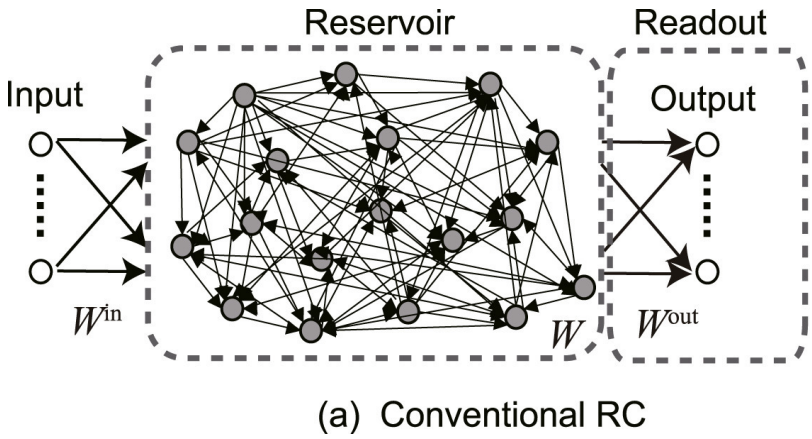

Figure 2.2: Structure of a Echo State Network. The image is from Tanaka et al. [92].

The reservoir consists of N nodes which are connected according a Erdős–Rényi graph model[19] [93]. 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 [94] to read more about the mathematical properties and how network is updated in detail.

[93]: Erdős et al. (1959)
[94]: Lukoševičius (2012)

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

20: Figure 2.2 uses upper case letter $\boldsymbol{W}$

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

3.1 Natural Intelligence

This thesis is inspired by the work "A Theory of Natural Intelligence' from von der Malsburg et al. [3]. Therefore, we dedicate this section to summarize their work in detail.

[3]: Malsburg et al. (2022)

According [3], 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 [95] 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 [96] 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 [97, 98]. 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 [99]. Thus, the brain develops as an overlay of attractor networks called net-fragments [100]. 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 nurture refers to the influence of experience and education

[95]: Mcpherson et al. (2001)

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

[96]: Kolmogorov (1998)

[97]: Willshaw et al. (1976)

[98]: Willshaw et al. (1979)

[99]: Malsburg et al. (1987)

[100]: Malsburg (2018)

Also, von der Malsburg et al. [3] discuss scene representation (i.e. how a scene is represented in the brain) even tough they point out that this is a contested concept [101]. 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 the framework of scene representation are the inductive bias that tunes the brain to the natural environment.

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

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. [3] 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 [102, 103]. They are composed of net-fragments that can be formed by network self-organization [104]. Ref. [3] 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

[102]: Arathorn (2002)
[103]: Olshausen et al. (1995)

[104]: Fernandes et al. (2015)

3.2 Self-Organization

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 [105]. DeepSwarm [106] 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, Fashion-MNIST, and CIFAR-10.

[105]: Dorigo et al. (1997)
[106]: 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 [107] or physical systems [108]. Neural Cellular Automata [109] 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[110]. NCAs can be trained efficiently with gradient descent to grow given 2D patterns such as images[111, 112]. 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 [114] use an NCA as decoder of a Variational Autoencoder [115]. This

[107]: Wolfram (1984)
[108]: Wolfram (1992)

[110]: Gilpin (2019)

[111]: Mordvintsev et al. (2020)
[112]: Mordvintsev et al. (2022)

4: a demo of this regeneration process is available at [113]

[114]: Palm et al. (2022)

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 [116] or generate structures with specific function such as simulated robots able to locomote[117]. Moreover, self-assembling approaches based on NCAs are not restricted to grid-structures. NCAs can be generalized to graph neural networks [118]. Graph cellular automata (GCA) use graph neural networks [119] 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. [120] propose to use NCA to classify given structures such as images. They apply the same network to each pixel 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. [121] 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 [122].

[116]: Sudhakaran et al. (2021)

[117]: Horibe et al. (2021)

[118]: Grattarola et al. (2021)

[119]: Zhou et al. (2021)

[120]: Randazzo et al. (2020)

[121]: Variengien et al. (2021)

[122]: 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[123]. 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.

[123]: 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" [124] uses the so called "ABCD" Hebbian learning rule which updates the weights as follows:

[124]: 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 [125] 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.

[125]: Kirsch et al. (2021)

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

[126]: Risi (2021)

Risi [126] 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).

3.2.1 Growing Networks

[127]: Kohonen (1982)
[128]: Kohonen (1989)

Unsupervised learning techniques usually map high dimensional input data to a lower-dimensional representation. One approach to do so are self-organizing maps (SOM) [127, 128]. They map the input data to a discretized representation of the input space of the training samples, 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.

[129]: Fritzke (1994)
[130]: Reilly et al. (1982)
[131]: Fritzke (1994)

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 [129–131] 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.

[132]: Marsland et al. (2002)

Grow When Required (GWR) networks [132] 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.

[133]: Mici et al. (2018)

Such GWR networks can be used to build self-organizing architectures. For example, Mici et al. [133] 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.

3.2.2 Self-Organization in Spiking Neural Networks

Spiking neural networks (SNNs) (c.f. Section Section 2.4.3) communicate through binary signals known as spikes and are very efficient on special event-based hardware[134]. 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.

[134]: Davies et al. (2018)

Similar to deep learning, there exists a multitude of different network architectures; Shallow [135, 136] and deep networks [137, 138] structures, fully connected [139] and convolutional layers [140, 141], as well as based on different learning rules such as supervised [142, 143], unsupervised [139, 144] and reinforcement learning based [145].

[135]: Kheradpisheh et al. (2018)
[136]: Mozafari et al. (2019)
[137]: Kheradpisheh et al. (2018)
[138]: Mozafari et al. (2018)
[139]: Kheradpisheh et al. (2018)
[140]: Diehl et al. (2015)
[141]: Tavanaei et al. (2016)
[142]: Diehl et al. (2015)
[143]: Zenke et al. (2018)
[144]: Ferré et al. (2018)
[145]: Mozafari et al. (2018)

A representable method for self-organization in SNNs is proposed by Raghavan et al. [146]. 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].

[146]: Raghavan et al. (2020)

6: for more information please refer to [146]

An alternative method proposed by Raghavan and Thomson [147] 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 [84] 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].

[147]: Raghavan et al. (2019)

[84]: Izhikevich (2003)

7: for more information please refer to [147]

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 [88, 148]. 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.

[148]: Vaila et al. (2019)
[88]: Kheradpisheh et al. (2018)

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

3.3 Visual Representation Learning

One of the earliest methods of learning visual representations are *Autoencoders*. *Autoencoders* have been introduced 1985 [149] and learn an encoder and a decoder function [150]. 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 [151]. 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. This constraint can be imposed with L_1 regularization or a Kullback-Leibler (KL) divergence between expected average neuron activation and an ideal distribution [152]. Other popular versions of *Autoencoders* are *Denoising Autoencoders* [153], *Contractive Autoencoders* [154], and *Variational Autoencoders (VAE)* [115]. 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. By doing so, the latent space becomes by design continuous and allows random sampling and interpolation of data.

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) [155]. This not only allows to generate part of images but also to learn visual representations of the image [156–158]. Other approaches rotate images and predict the rotation angle to generate representations [159, 160]. There exist many other methods that augment images and learn good visual representations based on it. A comprehensive overview is given by [161].

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

[149]: Rumelhart et al. (1985)
[150]: Baldi (2012)
[151]: Ranzato et al. (2007)
[152]: Le et al. (2012)
[153]: Vincent et al. (2008)
[154]: Rifai et al. (2011)
[155]: Kingma et al. (2014)
[156]: Pathak et al. (2016)
[157]: He et al. (2022)
[158]: Shi et al. (2022)
[159]: Komodakis et al. (2018)
[160]: Zhai et al. (2019)
[161]: Chen et al. (2022)

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 [26, 28, 162]. 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. [163] 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.

[26]: Chen et al. (2020)
[162]: He et al. (2020)
[28]: Caron et al. (2020)
[163]: Khosla et al. (2020)

3.4 Correlation within CNNs

TODO: Not sure if this chapter is still relevant.... -> move or delete?

Self-organization in neural networks can be done based on the input data. If Hebbian learning is used[9], cells are connected based on their correlation (i.e. cells with a high correlation are wired together). One way to capture the correlation within CNNs are Gram matrices. Gram matrices are essentially the dot-product between the channels of a feature map and can capture the style of given image. They are for example used for image style transfer[164] or related fields such as texture synthesis [165]. Appendix Chapter A provides an intuitive explanation what image style transfer is and how it is related to Gram matrices.

9: "Cells that fire together wire together"

[164]: Gatys et al. (2015)
[165]: Gatys et al. (2015)

A Gram matrix can be calculated based on the output of a convolutional layer. Each filter of a convolutional layer (i.e. each channel) produces a so called convolutional map. A convolutional map contains information about the content of the image such as object structure and positioning as well as information about the style. Calculating a Gram matrix eliminates content-related information from the convolutional layer output but does not affect style information (c.f. Appendix Chapter A). A Gram matrix calculates the correlations between the convolutional maps (i.e. between the filter responses) of a convolutional layer output. For a convolutional filter output F of layer l and two flattened convolutional maps i and j it is defined as [166]:

[166]: Gatys et al. (2016)

$$G^l_{ij} = \sum_k F^l_{ik} \cdot F^l_{jk} \tag{3.2}$$

Thereby, k is a hyperparameter defining how many elements of the convolutional output F are compared. Gatys et al. [166] applied this formula the first time to convolutional filters but did not fully explain why it works. However, they found that the style is captured well in the correlation between convolutional maps. Later, it was shown [167] that matching the Gram matrices between two convolutional filters can be reformulated as minimizing the Maximum Mean Discrepancy (MMD) [168] and thus that the style information is intrinsically represented by the distribution of activations in a CNN.

[166]: Gatys et al. (2016)
[167]: Li et al. (2017)
[168]: Gretton et al. (2012)

Intuitively, several filters together can describe the style of the image. For example, if one filter reacts to vertical white and black lines and a second filter reacts to horizontal white and black lines and the input image has a

checkerboard style, then these two filters have a high correlation, which is reflected in the Gram Matrix (c.f. Appendix Chapter A).

When Hebbian learning is used for self-organization, neurons of filters that often trigger together are connected. A dataset usually contains specific patterns, which are represented in the Gram Matrix[10]. Thus, neurons are connected that alone represent a certain filter but together represent a certain more complex pattern.

10: In the following the term pattern is used instead of style, because the Gram matrix can represent not only styles like photorealistic images, drawings, etc., but any non-content related information. For example, for an animal dataset, one filter could have high activation on white color, a second filter on black vertical lines, and a third filter on black dots. For a photo of a zebra, the first and second filters would have a high correlation while for a dalmatian, the first and third filters would have a high correlation

3.5 Leveraging Neuroscience for Deep Learning Based Object Recognition

TODO: beschreiben, dass diese Arbeit vom Natural Intelligence Paper inspiriert war

[169]: Lehmann (2022)

Prior to this thesis, Lehmann [169] examined self-organizing deep learning architectures in Master thesis at the Zurich University of Applied Sciences. 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 2.4.1). Similar to a convolutional filter, the convolutional feature map is calculated. Afterwards, the convolutional feature maps are compared and the lateral impact between the feature maps is calculated (i.e. how strongly the feature maps influence each other based on their covariance). When two feature maps are simultaneously 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.

Lehmann found that LCL layers increase robustness for object recognition. He shows on the MNIST dataset 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.

3.6 Rotation Invariant Convolutions

TODO: Not sure if this chapter is still relevant.... -> move or delete?

In this thesis, I show the capability of self-organization to deal with object transformations. Common transformations of objects in visual scenes are translations, rotations, zooming, and deformations. While most state-of-the-art architectures are translation invariant[11], they suffer from the other aforementioned transformations. The model can be trained to be more robust against most kind of transformations through data augmentation [171]. For example, dynamically increasing and decreasing image sizes works very well to obtain robustness against different object sizes. Robustness against deformations, on the other hand, can be learned to a great extend through data pre-processing such as random grid-based deformations [21, 172]. However, deformations are often domain specific. In this thesis, I mainly focus on rotation invariance of visual perception systems as this is a challenging task and is for many applications the most important transformation invariance beyond translation. Although

11: STOA architectures for visual scene interpretation are mainly based on convolutional neural networks and vision transformers [170]. These architectures are by definition translation invariant.

[171]: Simard et al. (2003)

[21]: Ronneberger et al. (2015)
[172]: Sager et al. (2022)

transformation invariant models can be obtain with data augmentation [171, 173], this approach is considered inefficient as the model has to learn many redundant parameters, independent for each rotation angle. In the following, we focus on methods that overcome this limitation.

A simple approach to achieve rotation invariance is to find the main axis of an image patch and to rotate it until it is aligned with the samples from the training set [174]. Another common strategy is to define features that are rotation invariant or equivariant, i.e. to use features whose output is either not affected by rotating the input image or whose output is rotated the same way as the input image by definition. Some well-known approaches are Local Binary Patterns [175], spiral resampling [176], and steerable pyramid filters [177].

[174]: Jafari-Khouzani et al. (2005)

[175]: Ojala et al. (2002)
[176]: Wu et al. (2008)
[177]: Greenspan et al. (1994)

Other approaches learn rotatable filters from the input data. Dieleman et al. [178] propose four new neural networks blocks. The probably most important block proposed in their work is a pooling operation that is applied over rotated feature maps to reduce the number of parameters and to learn rotation invariance more explicitly. Another approach [179] also applies convolutional filters to rotated versions of the image but aggregates the result by taking the maximum activation over the feature maps as output.

[178]: Dieleman et al. (2016)

[179]: Laptev et al. (2016)

Another category of approaches apply rotations to learned convolutional filters. Earlier approaches [180–182] use a Convolutional Restricted Boltzmann Machine (C-RBM)[12] [183] to tie the weights. Besides using C-RBM, it is also possible to tie the weights within several layers of a CNN to enforce rotation invariance and to reduce the number of parameters to learn. Teney and Herbert [184] split the filters of a CNN in orientation groups and constraint their weights. Such models achieve rotation covariance[13] and only need to learn a single canonical filter per orientation group. This concept can also be applied to the rotation group in the final layers of a CNN to obtain invariance to global rotations [185].

[180]: Schmidt et al. (2012)
[181]: Kivinen et al. (2011)
[182]: Sohn et al. (2012)
[183]: Le Roux et al. (2008)
[184]: Teney et al. (2016)
[185]: Wu et al. (2015)

12: A C-RBM is a generative stochastic network that can learn a probability distribution over its inputs. Multiple layers of C-RBM are also known as deep belief networks.

13: rotation covariance means that applying a rotation to the input image results in a shift of the output across the features

TODO: somewhere it is argued that biologically-motivated algorithms work bad: This is shown here: https://arxiv.org/pdf/1807.04587.pdf -> add this as source -> [13]

[13]: Bartunov et al. (2018)

Preliminary Study 4

In this preliminary study, I examine how representations of typical deep learning architectures look like. Especially net fragments are of interest, i.e. (groups of) neurons that represent certain (parts of) objects. According to [3], net fragments are a compositional data structure (c.f. Section 3.1), meaning that net fragments in the last layers are complexer and are composed of simpler fragments of previous layers. Thus, some (group) of neurons in the first layer should be active if some low-level features (e.g. part of objects) are present in the input. If several groups of such neurons are active, these neurons activate some other neurons in subsequent layers that represent higher-level net fragments.

[3]: Malsburg et al. (2022)

A typical task in the field of image analysis is image classification (cf. Section 6). 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 net fragment that represents 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 less complex objects that are composed by the last layer.

Deep Learning architectures usually consist of several layers with millions of neurons, leading to millions of activations [21, 41–45]. In addition, 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 task is proposed; The dataset consists of 10 images as shown in Figure 4.1. 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).

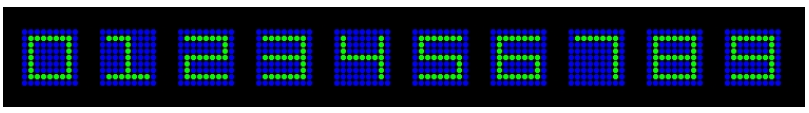

Figure 4.1: A novel dataset created to investigate the properties of modern deep learning architectures. The dataset consists of images of the numbers $0-9$, each image has a size of 9×9 pixels. The blue dots represent pixels with the value 0, green dots represent pixels with the value 1.

The goal of this task is to predict the number (labels $0-9$) that is shown in the image. This task is neither for humans nor deep learning architectures challenging. However, it is simple enough to investigate the activations of neural networks and to search for net fragments.

The following example explains how net fragments could theoretically be built in this data set. In total, the data set consists of 9 basic lines, that can be composed to 10 different digits. A low-level net fragment could represent such a basic line and be composed to higher-level net

fragments such as digits. Figure 4.2 visualizes these components (basic lines on the left, digits on the right).

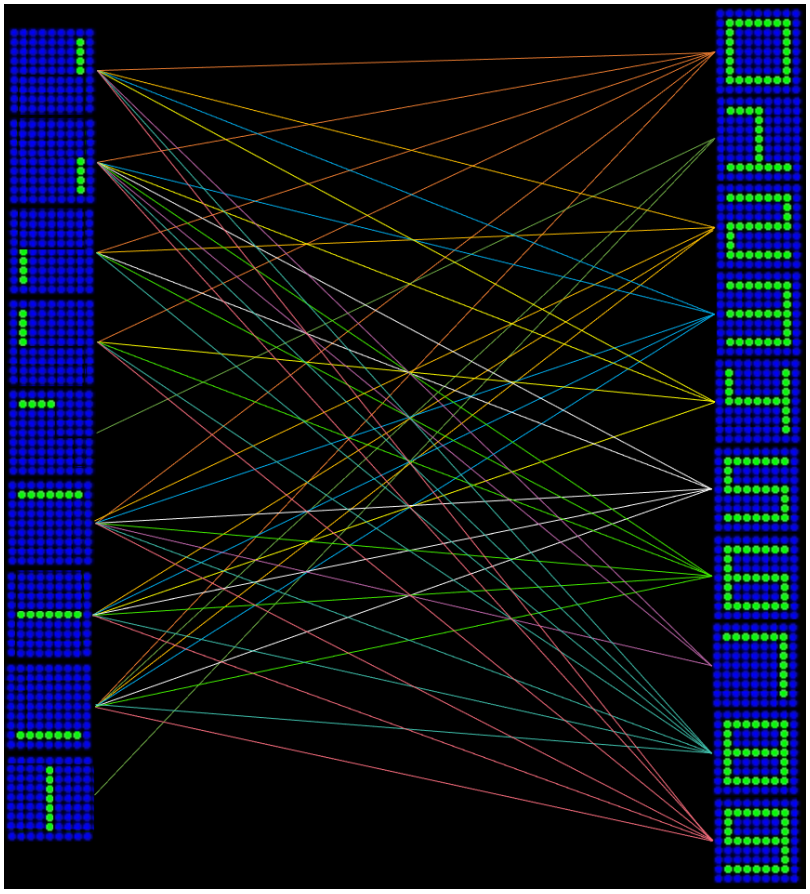

Figure 4.2: A visualisation of all the lines (left) needed to compose the digits in the data set (right). The coloured lines in-between illustrate the relationship between the lines and the digits.

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 4.2 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 4.3. In this example, it is demonstrated that a fragment representing a digit can be composed of other fragments that also represent digits. However, net fragments do not necessarily have to represent digits as shown in this example but can be a composition of multiple pixels without semantic.

The fragments described so far are to be understood as an explanatory example. A neural network might extract features in the first layers that do not correspond to vertical and horizontal lines and subsequently compose those features into more complex net fragments that are not as simple-to-interpret. Meaning that looking for neurons of a (for us humans) simple to interpret meaning might be hard. 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

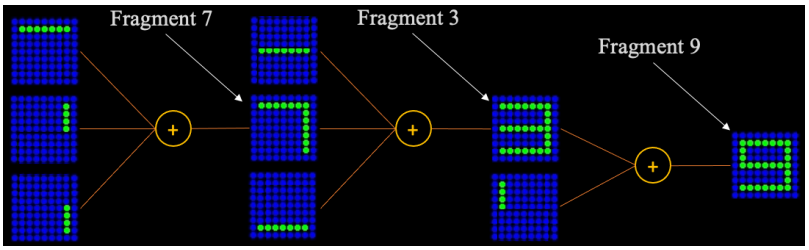

Figure 4.3: A sample composition of the net fragment that could represent the digit 9.

should only be active for digits with specific characteristics. Furthermore, digits have common traits; for example, the 7 is contained in the 3, the 3 is contained in the 9 and the 9 is contained in the 8. The 1, on the other hand, has nothing in common with the 4, etc. This means that at least one net fragment should be active for the 7, 3, 9, and 8 but not all net fragments that are composed to the digit 8 should be active if the input depicts a 9, 3, or 7. The net fragments between the 1 and 4, on the other hand, should be completely different. These properties should be visible in the activations of the network if it forms proper net fragments and are analysed in the following.

4.1 Methods

To investigate the emergence of net fragments, different deep learning models are trained. The models are trained on a supervised classification task to predict the corresponding class of the images of the dataset shown in Figure 4.1. The model's parameters are optimised by minimising the cross-entropy loss with the Adam optimizer [32]. 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.

[32]: 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" consists of 12 neurons. The number of neurons was 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 (12 neurons) $\rightarrow$ ReLU $\rightarrow$ FC (10 neurons) $\rightarrow$ Softmax.

After each epoch, the model's weights are stored as well as the activations

Table 4.1: A description of the different models used in the preliminary study.

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 (9 ∗ 9 neurons) → ReLU → head
11	Hand Crafted Conv. Layer for vertical & horizontal edge detection (kernel size=3 × 3, channels=2) → ReLU → head

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).

4.2 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. Interested readers who want to investigate the activations from all layers, as well as the network weights by themselves, can view an interactive visualisation at http://160.85.252.38:8501.

1: convolutional layers only consider a local neighbourhood by sliding a kernel over the input (c.f. Section 2.1.1)

Figure 4.4 shows the activations of the first fully connected layer in the "head" of the different models. The FC layer has 12 neurons whose acti-

vation 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.

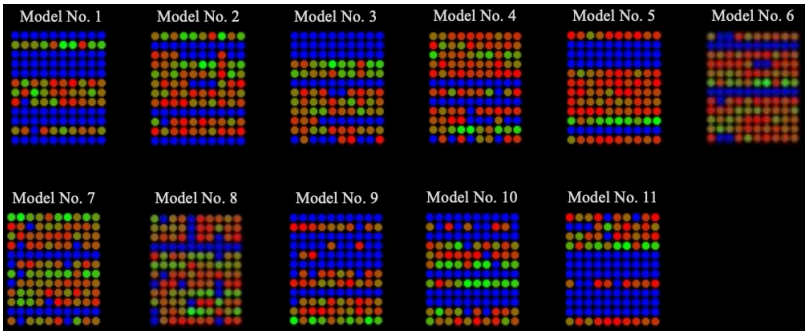

Figure 4.4: The activations of the first fully connected layer in the "head" of the different networks. For each model, the activity of the 12 neurons is shown for each class (activations along the vertical axis, classes along the horizontal axis). Red means low activation, green means high activation, blue means activation is off (i.e. 0).

Figure 4.5 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.

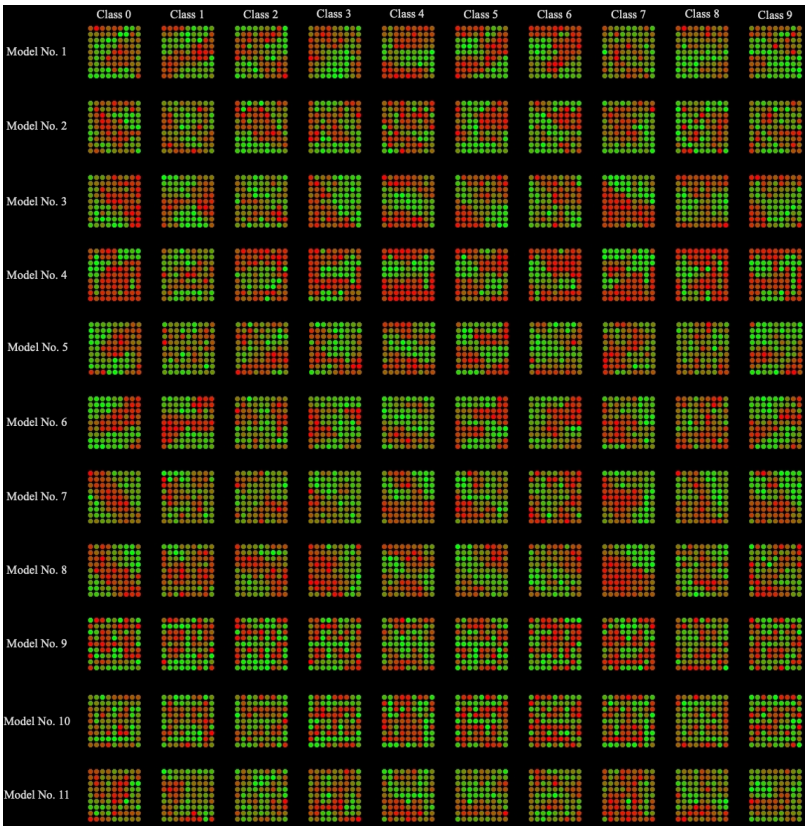

Figure 4.5: The inputs that maximize the probability that the different models predict a specific class. For example, the image in the top left corner maximizes the probability that model number 1 predicts that this is class 0 (digit 0).

4.3 Conclusion

Different architectures have been trained on a very simple and therefore easily interpretable data set. It was found that certain neurons are never active, while the active neurons usually remain active regardless of the class of the input data and only change their activity slightly. Net fragments, on the other hand, rely on being 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. Therefore, I argue that deep learning architectures do not comprise net fragments by default.

This assumption is further strengthened by the visualisation of the input that maximises the probability per class. If a class is represented by a net fragment and this net fragment is composed of other net fragments, then some of these lower-level fragments would have to be active. Such a set of low-level net fragments can directly be mapped to a set of pixels that has to be active. However, since the input does not contain a meaningful (sub-) set of active pixels, it can be assumed that these lower-level fragments are either not present or at least do not represent any meaningful features.

5 Future Work

5.1 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 [186]. 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. [187] 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.

[186]: Keller et al. (2012)

[187]: 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 [188]. 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.

[188]: LeCun (2022)

[187]: Keurti et al. (2022)

TODO: Write more about Paper from Grewe [187]

TODO: Stand jetzt sind es eher neue Module wie lateral connections -> gute Repräsentationen benötigen aber neue Architekturen, z.B. eine Hierarchie (Tiere -> Säugetiere, Reptilien, -> ...) und eine Menge an Actions, die jedes Element in der Hierarchie ausführen kann (z.B. drehen kann jedes Tier, Gehen nur die Landtiere und Schreiben nur der Mensch) -> Schreibe konkrete Vorschläge für Folgearbeiten!

Appendix

Image Style Transfer A

In the field of image style transfer[164], such correlations are used to generate images. An image is created based on two images, one image providing the content (i.e. the "content-image") and the other image providing the style (i.e. the "style-image"). To generate the content of the image, a random image can be fed into the model and a simple distance-based loss function such as the Euclidean norm can be used to calculate the difference between the model output and the content-image. Over time, the model would learn to output the content-image. However, only the content from the content-image and not its style is needed. A solution to this problem is using Convolutional feature maps[1] as they capture spatial information of an image well, while containing only little style information. Therefore, the distance loss is not calculated between the content-image and the model output but between the feature maps of the content-image and the output image.

[164]: Gatys et al. (2015)

1: a convolutional layer can have multiple filters (i.e. channels), each filter produces a convolutional feature map

The second task is to transfer the style from the style-image to the model output. The style of an image can be described by correlations across the different feature maps. This information about the image style can be calculated with a Gram matrix. A Gram matrix is the dot product[2] of the flattened feature vectors from a convolutional feature map (i.e. the dot product between all channels of a convolutional layer's output). For example, a convolutional layer could have multiple channels. The first channel could have a high activation for black and white stripes in horizontal direction while the second channel could have a high activation for black and white stripes in vertical direction. If both channels activate together for the same input and thus have a high correlation, there is a high possibility that the image might contain the style of a chessboard (i.e. black and white checkered). A third channel, for example, could have a high activation for eyes. If this channel has a low correlation with the first channel but a high correlation with the second channel (i.e. black and white stripes in vertical direction), the input image might contain the face of a zebra and thus capture a "zebra-style". Similar to the content-transfer, a distance loss such as the Euclidean distance can be used to compare the Gram matrices of the style-image and the output-image. Thus, it is compared if the output-image has the same style as the style-image.

2: the dot product between two vectors can be seen as similarity metric; it gets bigger if two vectors are more similar

In summary, the content is transferred by comparing entire convolutional layer output and the style by comparing the correlations between the feature maps of a convolutional layer output (i.e. by comparing gram matrices).

Design Decisions B

There exists a variety of possibilities how self-organization of the network architecture can be implemented. However, this thesis is of course limited in time and resources an thus not all of them can be investigated. Therefore, a promising direction of research had to be defined at the beginning and followed afterwards.

As a constraint it was defined that (i) a network without dynamics is developed and (ii) self-organization takes place across multiple layers. A network without dynamics is used because Deep Learning works very well for pattern recognition and the data analyzed by the computer is mostly static. Dynamic networks usually have poorer performance and use special algorithms to convert static data such as images into dynamic time-related signals (c.f. Section Section 3.2.2). Applying self-organization across multiple layers rather than just adding lateral connections in one layer is preferred as this allows the network to form more complex structures and thus to use the whole potential of self-organisation.

This thesis is strongly inspired by the paper "Natural Intelligence" [3] (c.f. Section Section 3.1). The most relevant findings in this work are:

[3]: Malsburg et al. (2022)

- The brain is highly structured
- Self-organization is the most promising approach to achieve natural intelligence. It loops between activity and connectivity
- Self-organization forms net-fragments, one neuron can be part of multiple fragments
  - An object can be represented by multiple net-fragments
  - A hierarchy of features can be represented by nested net-fragments
- The self-organizing process has to start from an already established coarse structure
- There exists many alternative pathways in the network

However, it is unclear how these findings and hypotheses from the field of neuroscience can be implemented in an algorithm. To create a self-organizing neural network, the following mechanisms are particularly relevant: The design of the initial architecture, the allowed architecture modifications (removing and adding neurons and/or connections), the type of neurons, and the definition of the learning algorithm. These aspects are discussed hereafter.

Design of initial architecture The initial architecture should exhibit a good inductive bias from the start so that the network can learn (something meaningful). Current approaches also start from much bigger architectures and reduce their size by up to a thirty-fold [124] (c.f. Section Section 3.2) or grow the architecture from a single neuron [147] (c.f. Section Section 3.2.2) by using self-organisation.

[124]: Pedersen et al. (2021)
[147]: Raghavan et al. (2019)

Architecture modifications The architecture can be modified in several ways. For example, from a more global perspective, the network can grow, shrink, split into sub-networks[1], or multiple sub-networks can re-organize into one network. From a local perspective, connections can be created or removed within the same layer (lateral connections) or between subsequent layer. Moreover, connections can be used to skip on or several layers (residual connections), or connections can be recurrent to a previous layer. Furthermore, besides modifying single connections and neurons, complete blocks of neurons can be added or removed. This includes for example layers, mathematical functions like pooling operations, or pre-defined modules consisting of several layers.

1: this is related to ensemble approaches TODO: add source

Type of neurons As described above, no dynamic (i.e. spiking) neurons are employed. The classical neurons multiply the input $\boldsymbol{x}$ by a weight vector $\boldsymbol{w}$, add a bias $\boldsymbol{b}$ and calculate the output with a non-linear activation of this sum (c.f. Section Section 2.1). However, recent architectures replace such classical neurons with more complex units. For example, Kirsch and Schmidhuber [125] (c.f. Section Section 3.2) use tiny RNNs with shared parameters as neurons. Such neurons are able to learn learning algorithms which are equivalent to back-propagation.

[125]: Kirsch et al. (2021)

Learning algorithm ... (see learning to learn or Hebbian based on input covarinace or learning based on target label (e.g. enforce some distinction in neurons covariance matrix depending on label))

Experiments Hebbian Learning

C

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 (2.9)) led to symmetric weights, which in turn led to poor representations. This symmetry could be broken by using the ABCD-Hebbian learning rule [189]. 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:

[189]: Niv et al. (2001)

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

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 [123], the coefficient were learnt through an evolution strategy [190]. However, this method does not scale for large networks with many parameters and did not achieve the desired performance in my experiments.

[123]: Najarro et al. (2020)
[190]: 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 C.1.

The image classification model is trained with backpropagation to minimize the negative log likelihood (NLL) loss. Adam [191] 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 $1 \cdot 10^{-4}$.

[191]: 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

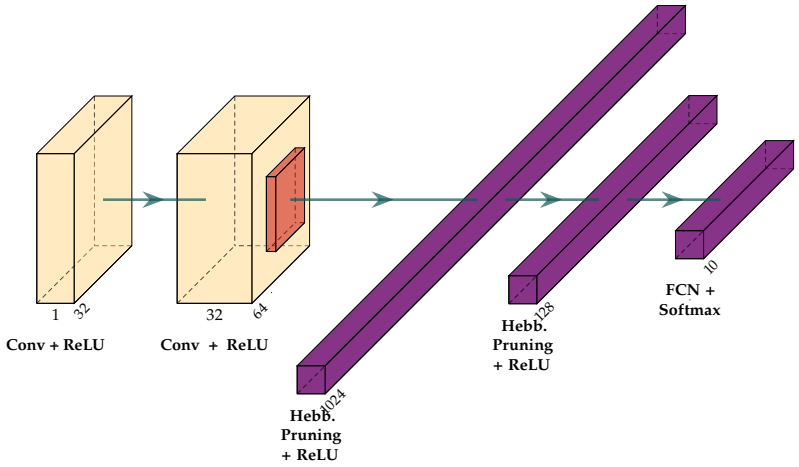

Figure C.1: The network architecture used to perform Hebbian-based weight pruning.

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:

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

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 [15] and evaluated on the MNIST as well as on the MNIST-C[1] dataset [192]. 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 C.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 [193]. Furthermore, it leads to sparse representations that can improve robustness. Figure Figure C.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.

[193]: Liang et al. (2021)

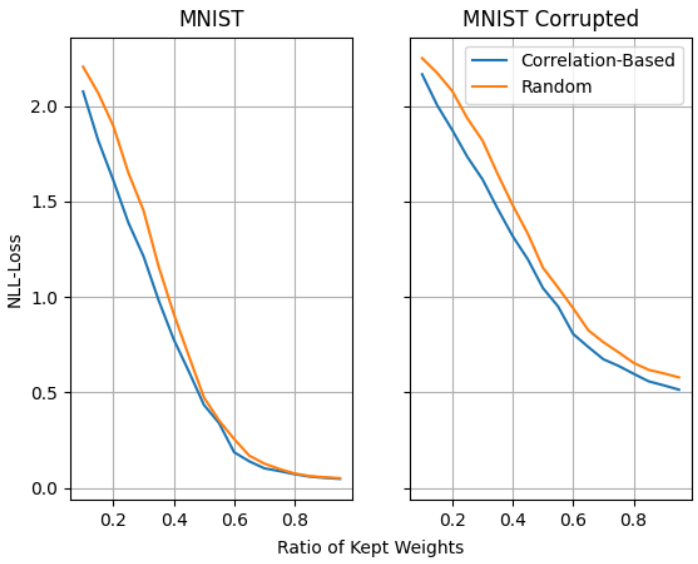

Figure C.2: The NLL loss of the *second* Hebbian based pruning method compared to removing random weights. The y axis shows the loss while the x axis shos the ratio of weights kept.

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.

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 2).

[2] JA Scott Kelso. *Dynamic patterns: The self-organization of brain and behavior*. MIT press, 1995 (cited on page 3).

[3] 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 3–5, 21, 22, 31, 43).

[4] Birgitta Dresp. 'Seven Properties of Self-Organization in the Human Brain'. In: *Big Data Cogn. Comput.* 4 (2020), p. 10 (cited on page 3).

[5] Francis Crick. 'The recent excitement about neural networks'. en. In: *Nature* 337.6203 (Jan. 1989), pp. 129–132. DOI: 10.1038/337129a0 (cited on page 3).

[6] 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 page 3).

[7] 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 3).

[8] 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 3).

[9] 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 3).

[10] 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 3).

[11] Geoffrey Hinton et al. 'How to do backpropagation in a brain'. In: *Invited talk at the NIPS'2007 deep learning workshop*. Vol. 656. 2007, pp. 1–16 (cited on page 3).

[12] Dong-Hyun Lee et al. 'Difference Target Propagation'. In: arXiv:1412.7525 (Nov. 2015). arXiv:1412.7525 [cs] (cited on page 3).

[13] 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 3, 29).

[14] 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 3).

[15] 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 3, 46).

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 'CIFAR-10 (Canadian Institute for Advanced Research)'. In: () (cited on page 3).

[17] 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 4).

[18] 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 4).

[19] Moheb Costandi. *Neuroplasticity*. The MIT Press essential knowledge series. Cambridge, MA: The MIT Press, 2016 (cited on page 4).

[20] 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 5).

[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 5, 10, 11, 28, 31).

[22] 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 5).

[23] 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 5).

[24] 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 5).

[25] 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 6).

[26] 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 6, 27).

[27] 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 6).

[28] 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 6, 27).

[29] 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 7).

[30] 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 7).

[31] 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 8).

[32] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *CoRR* abs/1412.6980 (2015) (cited on pages 9, 33).

[33] 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 9).

[34] Coursera Inc. *Deep Learning Specialization*. 2022. URL: https://www.coursera.org/specializations/deep-learning (visited on 08/19/2022) (cited on page 9).

[35] 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 10).

[36] 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 10).

[37] 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 page 10).

[38] 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 10, 11).

[39] 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 10, 11).

[40] 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 10, 11).

[41] Christian Szegedy et al. 'Going Deeper with Convolutions'. In: arXiv:1409.4842 (Sept. 2014). arXiv:1409.4842 [cs] (cited on pages 10, 11, 31).

[42] 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 10, 11, 31).

[43] 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 10, 11, 31).

[44] 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 10, 11, 31).

[45] 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 10, 11, 31).

[46] 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 11).

[47] 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 11).

[48] 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 11).

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

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

[51] 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 12).

[52] 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 12).

[53] 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 12).

[54] 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 12).

[55] 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 12).

[56] 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 12).

[57] 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 12).

[58] Yu Zhang and Qiang Yang. 'A Survey on Multi-Task Learning'. In: arXiv:1707.08114 (Mar. 2021). arXiv:1707.08114 [cs] (cited on page 12).

[59] 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 12).

[60] 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 12).

[61] 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 12).

[62] Gary Marcus. 'Deep Learning: A Critical Appraisal'. In: arXiv:1801.00631 (Jan. 2018). arXiv:1801.00631 [cs, stat] (cited on page 12).

[63] Hans Moravec. *Mind children: the future of robot and human intelligence*. eng. 4. print. Cambridge: Harvard Univ. Press, 1995 (cited on page 13).

[64] 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).

[65] 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 page 14).

[66] 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 14).

[67] 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 15).

[68] 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 15).

[69] 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 15).

[70] 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 15).

[71] 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 15).

[72] 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 15).

[73] Michael Teichmann and Fred Hamker. 'Intrinsic plasticity: A simple mechanism to stabilize Hebbian learning in multilayer neural networks.' In: Mar. 2015 (cited on page 15).

[74] 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 16).

[75] 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 16, 17).

[76] 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 16).

[77] Jason Weston, Sumit Chopra, and Antoine Bordes. 'Memory Networks'. In: arXiv:1410.3916 (Nov. 2015). arXiv:1410.3916 [cs, stat] (cited on page 16).

[78] 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 17).

[79] 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 17).

[80] 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 17).

[81] 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 17).

[82] Hubert Ramsauer et al. 'Hopfield Networks is All You Need'. In: arXiv:2008.02217 (Apr. 2021). arXiv:2008.02217 [cs, stat] (cited on page 17).

[83] 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 18).

[84] 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 18, 25).

[85] 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 18).

[86] Hélène Paugam-Moisy. 'Spiking Neuron Networks A survey'. In: (2006) (cited on page 18).

[87] 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 18).

[88] 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 18, 25).

[89] 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 19).

[90] 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 19).

[91] 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 19).

[92] 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 19).

[93] P. Erdös and A. Rényi. 'On Random Graphs I'. In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290 (cited on page 19).

[94] 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 19).

[95] 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).

[96] 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).

[97] 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).

[98] 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).

[99] 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).

[100] 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).

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

[102] 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).

[103] 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).

[104] 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).

[105] 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 22).

[106] 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 22).

[107] Stephen Wolfram. 'Cellular automata as models of complexity'. In: *Nature* 311.5985 (Oct. 1984), pp. 419–424. DOI: 10.1038/311419a0 (cited on page 22).

[108] 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 22).

[109] N. H. Wulff and John A. Hertz. 'Learning Cellular Automation Dynamics with Neural Networks'. In: *NIPS*. 1992 (cited on page 22).

[110] 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 22).

[111] Alexander Mordvintsev et al. 'Growing Neural Cellular Automata'. In: *Distill* (2020) (cited on page 22).

[112] 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 22).

[113] Distill. *Growing Neural Cellular Automata*. 2022. URL: https://distill.pub/2020/growing-ca (visited on 09/05/2022) (cited on page 22).

[114] Rasmus Berg Palm et al. 'Variational Neural Cellular Automata'. In: arXiv:2201.12360 (Feb. 2022). arXiv:2201.12360 [cs] (cited on page 22).

[115] Diederik P. Kingma and Max Welling. 'Auto-Encoding Variational Bayes'. In: arXiv:1312.6114 (May 2014). arXiv:1312.6114 [cs, stat] (cited on pages 22, 26).

[116] 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).

[117] 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).

[118] 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).

[119] 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).

[120] 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).

[121] 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 23).

[122] Volodymyr Mnih et al. 'Playing Atari with Deep Reinforcement Learning'. In: arXiv:1312.5602 (Dec. 2013). arXiv:1312.5602 [cs] (cited on page 23).

[123] 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 23, 45).

[124] 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 pages 23, 43).

[125] Louis Kirsch and Jürgen Schmidhuber. 'Meta Learning Backpropagation And Improving It'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021 (cited on pages 23, 44).

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

[127] 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).

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

[129] 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 24).

[130] 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 24).

[131] 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 24).

[132] 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 24).

[133] 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 24).

[134] 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).

[135] 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).

[136] 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).

[137] 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).

[138] 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).

[139] 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).

[140] 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).

[141] 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).

[142] 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).

[143] 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).

[144] 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).

[145] 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).

[146] 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 25).

[147] Guruprasad Raghavan and Matt Thomson. 'Neural networks grown and self-organized by noise'. In: *NeurIPS*. 2019 (cited on pages 25, 43).

[148] 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 25).

[149] 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 page 26).

[150] 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, Feb. 2012, pp. 37–49 (cited on page 26).

[151] 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 26).

[152] 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 page 26).

[153] 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 26).

[154] 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 26).

[155] 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 26).

[156] 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 26).

[157] 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 26).

[158] Yuge Shi et al. 'Adversarial masking for self-supervised learning'. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20026–20040 (cited on page 26).

[159] Nikos Komodakis and Spyros Gidaris. 'Unsupervised representation learning by predicting image rotations'. In: *International Conference on Learning Representations (ICLR)*. 2018 (cited on page 26).

[160] Xiaohua Zhai et al. 'S4L: Self-Supervised Semi-Supervised Learning'. In: arXiv:1905.03670 (July 2019). arXiv:1905.03670 [cs] (cited on page 26).

[161] 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 26).

[162] 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 27).

[163] Prannay Khosla et al. 'Supervised contrastive learning'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18661–18673 (cited on page 27).

[164] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 'A Neural Algorithm of Artistic Style'. In: arXiv:1508.06576 (Sept. 2015). arXiv:1508.06576 [cs, q-bio] (cited on pages 27, 41).

[165] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 'Texture Synthesis Using Convolutional Neural Networks'. In: arXiv:1505.07376 (Nov. 2015). arXiv:1505.07376 [cs, q-bio] (cited on page 27).

[166] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 'Image Style Transfer Using Convolutional Neural Networks'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2414–2423. DOI: 10.1109/CVPR.2016.265 (cited on page 27).

[167] Yanghao Li et al. 'Demystifying Neural Style Transfer'. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. Melbourne, Australia: AAAI Press, 2017, pp. 2230–2236 (cited on page 27).

[168] Arthur Gretton et al. 'A Kernel Two-Sample Test'. In: *Journal of Machine Learning Research* 13.25 (2012), pp. 723–773 (cited on page 27).

[169] Claude Lehmann. 'Leveraging Neuroscience for Deep Learning Based Object Recognition'. MA thesis. Zurich University of Applied Sciences, 2022 (cited on page 28).

[170] Alexey Dosovitskiy et al. 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale'. In: arXiv:2010.11929 (June 2021). arXiv:2010.11929 [cs] (cited on page 28).

[171] P.Y. Simard, D. Steinkraus, and J.C. Platt. 'Best practices for convolutional neural networks applied to visual document analysis'. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* Vol. 1. Edinburgh, UK: IEEE Comput. Soc, 2003, pp. 958–963. DOI: 10.1109/ICDAR.2003.1227801 (cited on pages 28, 29).

[172] Pascal Sager et al. 'Unsupervised Domain Adaptation for Vertebrae Detection and Identification in 3D CT Volumes Using a Domain Sanity Loss'. en. In: *Journal of Imaging* 8.8 (Aug. 2022), p. 222. DOI: 10.3390/jimaging8080222 (cited on page 28).

[173] B. Fasel and D. Gatica-Perez. 'Rotation-Invariant Neoperceptron'. In: *18th International Conference on Pattern Recognition (ICPR'06)*. Hong Kong, China: IEEE, 2006, pp. 336–339. DOI: 10.1109/ICPR.2006.1020 (cited on page 29).

[174] K. Jafari-Khouzani and H. Soltanian-Zadeh. 'Radon transform orientation estimation for rotation invariant texture analysis'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.6 (June 2005), pp. 1004–1008. DOI: 10.1109/TPAMI.2005.126 (cited on page 29).

[175] T. Ojala, M. Pietikainen, and T. Maenpaa. 'Multiresolution gray-scale and rotation invariant texture classification with local binary patterns'. en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (July 2002), pp. 971–987. DOI: 10.1109/TPAMI.2002.1017623 (cited on page 29).

[176] Wen-Rong Wu and Shieh-Chung Wei. 'Rotation and gray-scale transform-invariant texture classification using spiral resampling, subband decomposition, and hidden Markov model'. In: *IEEE Transactions on Image Processing* 5.10 (Oct. 1996), pp. 1423–1434. DOI: 10.1109/83.536891 (cited on page 29).

[177] Hayit Greenspan et al. 'Rotation invariant texture recognition using a steerable pyramid'. In: *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*. Vol. 2. IEEE. 1994, pp. 162–167 (cited on page 29).

[178] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. 'Exploiting Cyclic Symmetry in Convolutional Neural Networks'. In: arXiv:1602.02660 (May 2016). arXiv:1602.02660 [cs] (cited on page 29).

[179] Dmitry Laptev et al. 'TI-POOLING: transformation-invariant pooling for feature learning in Convolutional Neural Networks'. In: arXiv:1604.06318 (Sept. 2016). arXiv:1604.06318 [cs] (cited on page 29).

[180] U. Schmidt and S. Roth. 'Learning rotation-aware features: From invariant priors to equivariant descriptors'. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. Providence, RI: IEEE, June 2012, pp. 2050–2057. DOI: 10.1109/CVPR.2012.6247909 (cited on page 29).

[181] Jyri J. Kivinen and Christopher K. I. Williams. 'Transformation Equivariant Boltzmann Machines'. In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela et al. Vol. 6791. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–9. DOI: 10.1007/978-3-642-21735-7_1 (cited on page 29).

[182] Kihyuk Sohn and Honglak Lee. 'Learning Invariant Representations with Local Transformations'. In: arXiv:1206.6418 (June 2012). arXiv:1206.6418 [cs, stat] (cited on page 29).

[183] Honglak Lee et al. 'Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations'. en. In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. Montreal, Quebec, Canada: ACM Press, 2009, pp. 1–8. DOI: 10.1145/1553374.1553453 (cited on page 29).

[184] Damien Teney and Martial Hebert. 'Learning to Extract Motion from Videos in Convolutional Neural Networks'. In: arXiv:1601.07532 (Jan. 2016). arXiv:1601.07532 [cs] (cited on page 29).

[185] Fa Wu, Peijun Hu, and Dexing Kong. 'Flip-Rotate-Pooling Convolution and Split Dropout on Convolution Neural Networks for Image Classification'. In: arXiv:1507.08754 (July 2015). arXiv:1507.08754 [cs] (cited on page 29).

[186] 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 page 37).

[187] 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 pages 37, 38).

[188] Yann LeCun. *A Path Towards Autonomous Machine Intelligence*. 2022. URL: https://openreview.net/forum?id=BZ5a1r-kVsf (visited on 09/21/2022) (cited on page 37).

[189] 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 45).

[190] 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 page 45).

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

[192] 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 46).

[193] 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 47).