

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Filip Rechtorík

Deep Neural Networks with Biologically Inspired Local Connectivity and Topographic Organization

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: Mgr. Ján Antolík, Ph.D.

Study programme: Computer Science - Artificial Intelligence

Prague 2026

I declare that I carried out this master thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to express my deep gratitude to my supervisor, Mgr. Ján Antolík, Ph.D., for his consistent guidance and support throughout this thesis. Although the thesis took longer than originally expected, he always found time to meet with me and provide valuable guidance. I would also like to thank RNDr. Milan Straka, Ph.D., for discussing certain aspects of my work and providing valuable feedback. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic. Without the vast resources provided by this infrastructure, the extensive experiments required for this work would not have been possible. Finally, I would like to thank the Typst team for creating a typesetting system that made preparing this thesis enjoyable.

Title: Deep Neural Networks with Biologically Inspired Local Connectivity and Topographic Organization

Author: Bc. Filip Rechtorík

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Ján Antolík, Ph.D., Department of Software and Computer Science Education

Abstract: Convolutional neural networks achieve strong computer vision performance, but their rigid weight sharing and channel-based feature organization are poorly aligned with the topographic organization of biological visual cortex. This thesis investigates whether cortical-like topographic organization can emerge from architectural constraints, rather than being directly imposed through a similarity loss. We introduce LocalLinear, a generalization of locally connected layers that maps between two-dimensional neuronal sheets of arbitrary sizes and densities, replacing convolutional weight sharing with localized, neuron-specific connectivity. To structure the resulting connections, weights are generated by hypernetworks conditioned on neuronal descriptors and spatial offsets. To make these models trainable, we implement a custom CUDA kernel and memory-efficient hypernetwork backpropagation. The best-performing hypernetwork-based variant achieves image classification performance comparable to FLOP-matched convolutional baselines, while sinusoidal implicit representations produce the clearest smooth orientation preference maps. Together, these results suggest a path toward deep visual models whose neuronal structure can be represented as a function over continuous space.

Keywords: biologically inspired deep learning, topographic organization, local connectivity, hypernetworks, implicit neural representations

Název práce: Hluboké neuronové sítě s biologicky inspirovanou lokální konektivitou a topografickou organizací

Autor: Bc. Filip Rechtorík

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Ján Antolík, Ph.D. Katedra softwaru a výuky informatiky

Abstrakt: Konvoluční neuronové sítě dosahují velmi dobrých výsledků v úlohách počítačového vidění, avšak jejich striktní sdílení vah a organizace reprezentací do kanálů neodpovídají topografické organizaci biologické zrakové kůry. Tato práce zkoumá, zda může kortikálně podobná topografická organizace vznikat z architektonických omezení, spíše než být přímo vynucena pomocí podobnostní ztrátové funkce. Představujeme vrstvu LocalLinear, zobecnění lokálně propojených vrstev, která umožňuje mapování mezi dvourozměrnými neuronálními plochami libovolných velikostí a hustot a nahrazuje konvoluční sdílení vah lokalizovanou, neuronově specifickou konektivitou. Pro strukturování výsledných spojení jsou váhy generovány hypernetworks podmíněnými neuronálními deskriptory a prostorovými posuny. Aby bylo možné tyto modely trénovat, implementujeme vlastní CUDA jádro a paměťově efektivní zpětnou propagaci pro hypernetworks. Nejlepší hypernetwork varianta dosahuje úspěšnosti v klasifikaci obrazů srovnatelné s konvolučními baseline modely odpovídajícími počtem FLOPs, zatímco implicitní neurální reprezentace se sinusoidálními aktivacemi vytvářejí nejkoherentnější mapy preferované orientace. Společně tyto výsledky naznačují cestu k hlubokým zrakovým modelům, jejichž neuronální struktura může být reprezentována jako funkce nad spojitým prostorem.

Klíčová slova: biologicky inspirované hluboké učení, topografická organizace, lokální konektivita, hypernetworks, implicitní neurální reprezentace

Contents

Introduction **7**

1 Biological Background **10**
1.1 Neuron . . . 10
1.1.1 Neuronal Membrane . . . 11
1.1.2 Action Potential . . . 12
1.1.3 Neurites and Synapses . . . 14
1.2 The Visual System . . . 15
1.2.1 Overview of the Visual Pathway . . . 15
1.2.2 Retinotopy . . . 17
1.2.3 Retina . . . 18
1.2.4 Lateral Geniculate Nucleus . . . 20
1.2.5 Primary Visual Cortex (V1) . . . 20
1.2.6 Orientation Selectivity and Maps in V1 . . . 21
1.2.7 Extrastriate Cortex . . . 24

2 Deep Learning Background **25**
2.1 Artificial Neuron . . . 25
2.2 Feedforward Neural Networks . . . 26
2.2.1 Probabilistic Interpretation . . . 27
2.3 Training Neural Networks . . . 27
2.3.1 Maximum Likelihood Estimation . . . 27
2.3.2 Minibatch SGD . . . 28
2.3.3 Gradient Computation . . . 28
2.4 Computer Vision . . . 30
2.4.1 2D Convolutional Layer . . . 30
2.4.2 Convolutional Architectures . . . 31
2.5 Hypernetworks . . . 33
2.6 Implicit Neural Representations . . . 33

3 Related Work **34**

4 Methods **35**
4.1 LocalLinear Layer . . . 35
4.1.1 Implementation . . . 36
4.1.2 Weight Initialization . . . 37
4.1.3 Custom CUDA Implementation . . . 38
4.2 Hypernetwork Parameterization . . . 39
4.2.1 Memory Optimization . . . 40
4.2.2 Learning-Rate Scaling of Neuronal Descriptors . . . 41
4.2.3 Hypernetwork Initialization . . . 41
4.3 SIREN-Based Parameterization . . . 41
4.4 Network Architecture . . . 43
4.4.1 LocalLinear Models . . . 43
4.4.2 Convolutional Baselines . . . 45
4.5 Parameter Counts and FLOPs . . . 46
4.6 Visualizations . . . 47
4.7 Dataset and Task . . . 48
4.8 Training Setup and Hyperparameters . . . 48

5 Results **49**
5.1 Connection Fields and Topography 49
5.2 Image Classification Performance 55
5.3 Computational Scaling 56
5.4 Memory Analysis 57

6 Discussion **60**

Bibliography **63**

List of Figures **67**

List of Tables **69**

A Preliminary Hyperparameter Selection **70**

B Attachments **71**

Introduction

While current deep learning methods achieve state-of-the-art performance in computer vision tasks such as image classification and object detection (He et al., 2015a; Krizhevsky et al., 2017), their architectures have diverged substantially from their original biological inspiration. This architectural gap makes it difficult to use deep learning models directly to study biological visual processing.

Convolutional neural networks (CNNs) remain among the most successful computer vision architectures and were themselves partly motivated by the hierarchical organization of the visual cortex (Fukushima, 1980; LeCun et al., 1989). However, their core design relies on rigid weight sharing through convolutional kernels, which apply the same feature detectors at every spatial position, with features arranged in discrete, independent channels. While computationally efficient, this differs from the topographic organization of the mammalian visual cortex, where neighboring cortical regions tend to represent either neighboring regions of visual space or similar neuronal response properties, such as preferred orientation (Bear et al., 2016, pp. 342–354). As a result, establishing a direct correspondence between artificial units and biological neurons remains challenging.

This architectural mismatch matters in both directions. For neuroscience, it limits the extent to which modern deep learning models can serve as interpretable and biologically plausible models of vision. For machine learning, the robustness of biological vision suggests that biological neural organization may contain useful architectural principles (Geirhos et al., 2017), but the mismatch between CNNs and cortical organization makes these principles difficult to translate into deep learning architectures. A closer alignment between artificial and biological neural organization could therefore benefit both computational neuroscience and deep learning.

The aim of this thesis is to develop and evaluate a novel deep learning framework that generalizes convolutional-style architectures from discrete channel-based feature representations toward spatially organized neuronal sheets, while more closely aligning them with the topographic organization observed in biological visual cortex. In particular, the thesis investigates whether the proposed architecture gives rise to structured connection fields at the level of individual neurons, such as oriented or Gabor-like fields, and whether these local properties become smoothly organized at the population level across neuronal sheets, for example in the form of orientation preference maps. Both levels of organization are examined as emergent consequences of the model's architectural constraints.

With this aim, the thesis addresses the following three research questions:

1. Can convolutional weight sharing be replaced by localized, neuron-specific connectivity while remaining trainable and computationally feasible?

2. How do different architectural components influence the emergence of structured connection fields and smooth population-level topographic organization in the proposed framework?

3. How do these architectural components affect image classification performance?

To answer these questions, the thesis develops a framework built around several key components. At its core is a new layer type, LocalLinear, which replaces convolutional weight sharing and channel-based feature organization with localized, neuron-specific connections over neuronal sheets. Unlike standard convolutional layers or typical locally connected layers, this formulation allows mappings between neuronal sheets of arbitrary size and resolution, enabling a more flexible and biologically aligned spatial organization.

Removing weight sharing alone, however, leads to unstructured and noisy connection fields. To mitigate this, the thesis introduces a parameterization mechanism that learns shared rules for generating connections across neuronal space. In particular, it develops a hypernetwork-based approach (Ha et al., 2016) in which each neuron is associated with a compact learnable descriptor, and connection weights are generated from pairs of such descriptors together with their relative spatial offsets. This allows structured connection fields to emerge while preserving architectural flexibility, and it makes connectivity more interpretable by expressing it through a low-dimensional, shared generative rule rather than independent weights.

While this significantly improves the connection fields of individual neurons, it is not sufficient to produce smooth topographic organization at the population level. The framework therefore further incorporates implicit neural representations with sinusoidal activations (Sitzmann et al., 2020). When applied to the hypernetworks, these improve both performance and parameter efficiency. Applied to the parameterization of neuronal sheets, they support the emergence of smooth topographic structure across all layers, including orientation preference maps.

Together, these components form a unified framework for studying how structured connection fields and population-level topographic organization can emerge in deep learning models under biologically inspired architectural constraints.

More broadly, the framework interprets connectivity and neuronal descriptors as functions over two-dimensional neuronal space rather than as values tied to a fixed grid, making it conceptually compatible with different neuronal densities and, in principle, nonuniform spatial sampling.

Overall, the results show that localized neuron-specific connectivity alone is insufficient to produce coherent topographic organization. However, when combined with shared hypernetwork-based connection rules and implicit neural representations with sinusoidal activations, the proposed framework gives rise to structured connection fields and smooth population-level topographic organization, including orientation preference maps. Furthermore, when compared with convolutional networks matched in FLOPs and depth, the proposed framework achieves comparable image classification performance.

1 Biological Background

1.1 Neuron

The brain is composed primarily of two types of cells: neurons and glia (Bear et al., 2016, p. 24). Whilst the main contribution of glial cells is to insulate, support, and nourish neurons, neurons themselves perform most of the information processing.

As described by Bear et al. (2016, pp. 29–46), every neuron consists of a cell body—the soma—as well as a number of neurites extending outward (Figure 1.1). These neurites are classified into two types: *dendrites*, serving as the primary source of input for the neuron, and a single *axon*, serving as the output carrier.

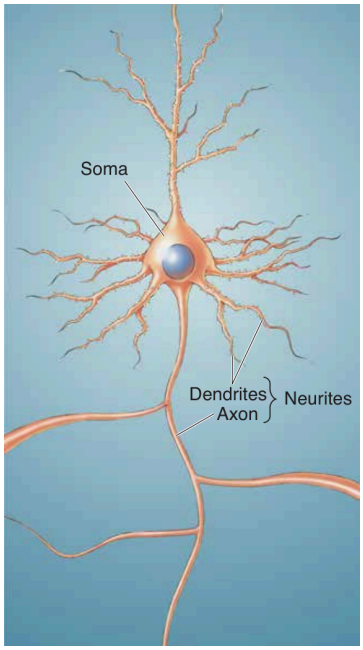


Figure 1.1: The basic parts of a neuron. Reprinted from Bear et al. (2016, p. 26).

The neuron, like most other cells in the body, is enclosed in a protective membrane, and contains a nucleus housing the DNA (Bear et al., 2016, pp. 29–38). It also possesses the typical cellular organelles—the "organs" of the cell—such as mitochondria for energy production and ribosomes for protein synthesis. The cellular structure of a typical neuron is illustrated in Figure 1.2.

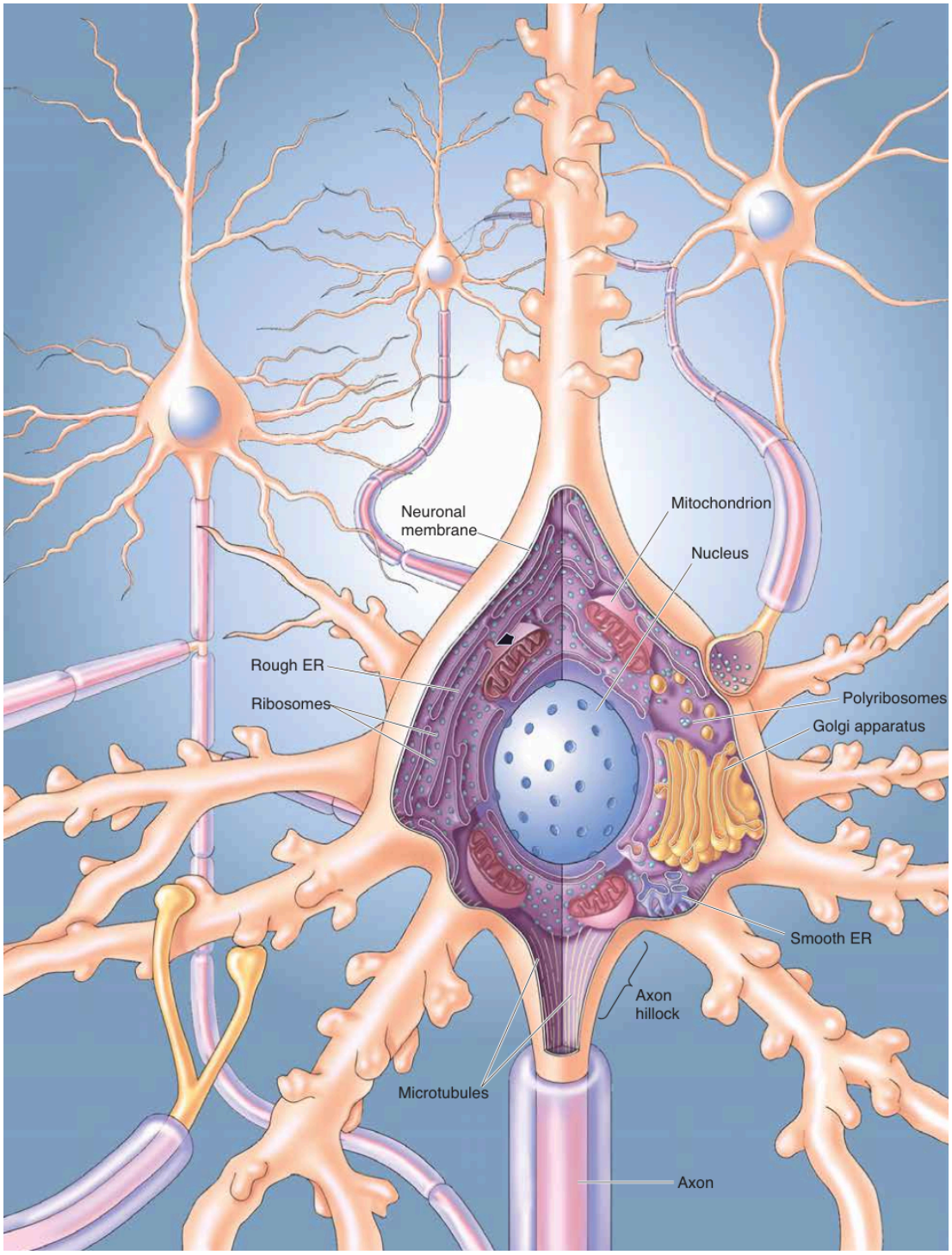


Figure 1.2: The internal structure of a typical neuron. Reprinted from Bear et al. (2016, p. 30).

1.1.1 Neuronal Membrane

The neuronal membrane—a phospholipid bilayer—separates the inside of the cell from the outside and plays a critical role in the ability of neurons to process information (Bear et al., 2016, pp. 56–70). By maintaining different concentrations of electrically charged ions on either side of the membrane, it creates an

electrical potential difference—called *membrane potential*—across the membrane. This is illustrated in Figure 1.3.

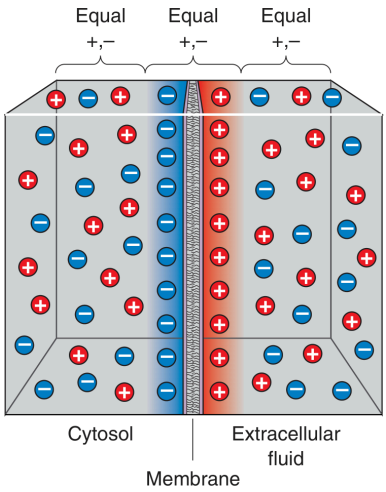


Figure 1.3: The uneven distribution of ions across the neuronal membrane, giving rise to the membrane potential. Reprinted from Bear et al. (2016, p. 68).

As described by Bear et al. (2016, pp. 62–64), the neuronal membrane, otherwise impermeable to ions, is filled with *ion channels*, which allow the selective passing of specific ions (typically K^+, Na^+, Cl^-, or Ca^{2+}) from one side to the other. The random movement of particles then makes it more likely that the ions will cross from the side with higher concentration to the side with lower concentration. This movement is called *diffusion*, and its strength depends on the number of ion channels as well as the *concentration gradient*. Some of these channels are always open, while others, like the *voltage-gated sodium channel* open conditionally.

The diffusion is complemented by the effects of the electrostatic forces induced by the electrically charged ions (Bear et al., 2016, pp. 64–70). Through these processes, the ionic composition eventually reaches an equilibrium in which the diffusion of ions exactly counterbalances the electrostatic forces, resulting in a *resting membrane potential*, typically equal to $-65\,\mathrm{mV}$.

1.1.2 Action Potential

Most neurons communicate through *action potentials*, or *spikes*, which can be thought of as discrete events of information transfer. Bear et al. (2016, p. 84) describes them as "all-or-none".

A neuronal membrane responds to an incoming action potential by either *depolarizing* or *hyperpolarizing*—the membrane potential becoming less or more negative, respectively (Bear et al., 2016, pp. 124–125). If the membrane depolarizes beyond a specific *threshold*, typically around $-40\,\mathrm{mV}$, it triggers a new action potential (Bear et al., 2016, pp. 82–84, 94).

This process utilizes voltage-gated sodium channels, which open in response to the initial depolarization, allowing Na^+ ions to flow in and cause a rapid chain reaction of further depolarization, travelling like a wave through the neuronal body and the axon, and communicating the information of action potential to

other neurons (Bear et al., 2016, pp. 88-90, 100). Once the peak depolarization is reached, the voltage-gated sodium channels inactivate and potassium channels open to release K^+ out of the cell, restoring the resting membrane potential[1]. The varying membrane potential during an action potential is illustrated in Figure 1.4.

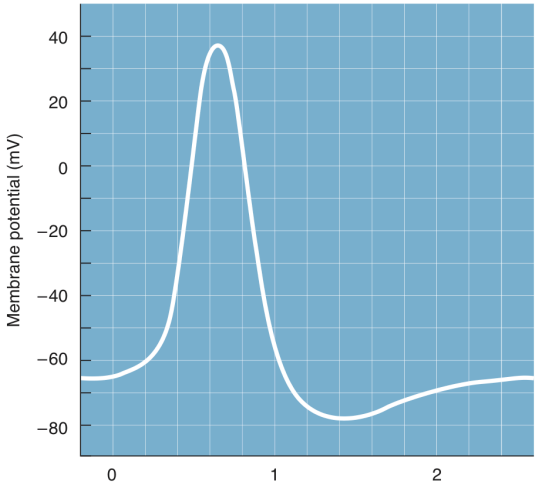


Figure 1.4: An action potential displayed by an oscilloscope. Adapted from Bear et al. (2016, p. 84).

As noted by Dayan & Abbott (2001, pp. 8–12), action potentials (spikes) convey information through their timing. A sequence of spikes emitted by a single neuron is called a *spike train*. Figure 1.5 shows an example recording of spike trains across multiple neurons. However, because of the randomness of these spike trains across trials, they are typically measured statistically as *firing rates*, representing the average number of spikes per unit of time.

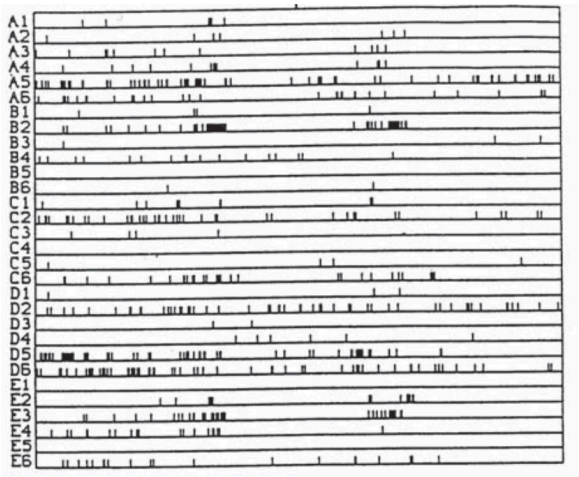


Figure 1.5: A spike train recording. Each row represents a different neuron and the x-axis represents time. Each vertical bar is a single spike. Reprinted from Gerstner & Kistler (2002, p. 23).

[1]To recover from this ionic exchange, *sodium-potassium pumps* utilize energy (ATP) to actively transfer Na^+ back out of the cell and K^+ back into the cell, maintaining the long-term concentration gradients (Bear et al., 2016, p. 88).

1.1.3 Neurites and Synapses

Axons are the specialized structures used by neurons to send their action potentials to other targets, with lengths ranging from less than a millimeter to over a meter (Bear et al., 2016, pp. 39–41). Each axon originates at the *axon hillock* of the neuron and can branch into many *collaterals*, allowing a single neuron to communicate with many different targets.

At the end of an axon lie the *axon terminals* (*terminal boutons*), which form connections with other neurons, typically on their dendrites or cell bodies (Bear et al., 2016, pp. 41–43). These connections, called *synapses*, are where action potentials are transmitted from one neuron to another. In most cases, this transmission is mediated through a chemical signal known as a *neurotransmitter*.

When an action potential arrives at an axon terminal, it releases neurotransmitter molecules that bind to the receiving neuron's *receptors* (Bear et al., 2016, pp. 113–124). After binding, *excitatory* neurotransmitters cause the neuronal membrane to depolarize, while *inhibitory* neurotransmitters cause it to hyperpolarize[2].

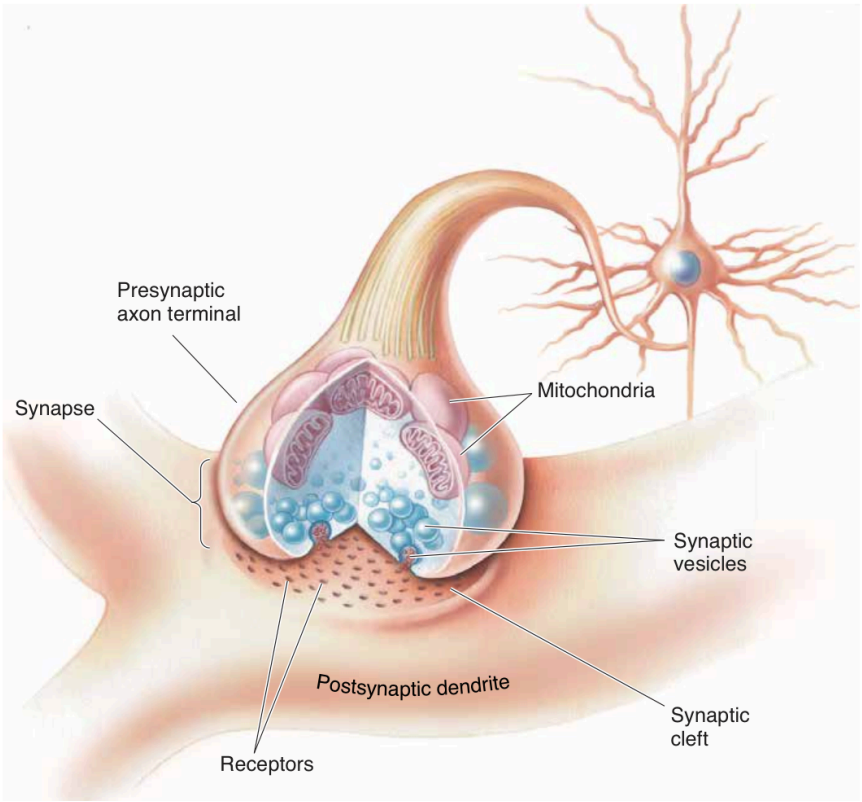


Figure 1.6: The axon terminal and the synapse. Reprinted from Bear et al. (2016, p. 42).

Interestingly, according to *Dale's Principle*—coined by Eccles et al. (1954)—each neuron releases the same neurotransmitter from all its terminals and is therefore either purely excitatory or purely inhibitory.

[2]The depolarization and hyperpolarization of the membrane is achieved by the neurotransmitter binding to *transmitter-gated ion channels*, which activate to allow the passage of selected ions, resulting in the change of membrane potential (Bear et al., 2016, p. 124).

1.2 The Visual System

1.2.1 Overview of the Visual Pathway

Visible light—consisting of electromagnetic waves in the wavelength range of approximately 400–700 nm—enters the eye (Figure 1.7) through the pupil and reaches the retina at the back of the eye (Bear et al., 2016, pp. 294–296).

The retina is covered with photoreceptors, which convert the electromagnetic radiation into changes in membrane potentials (Bear et al., 2016, p. 312). This process is called *phototransduction*.

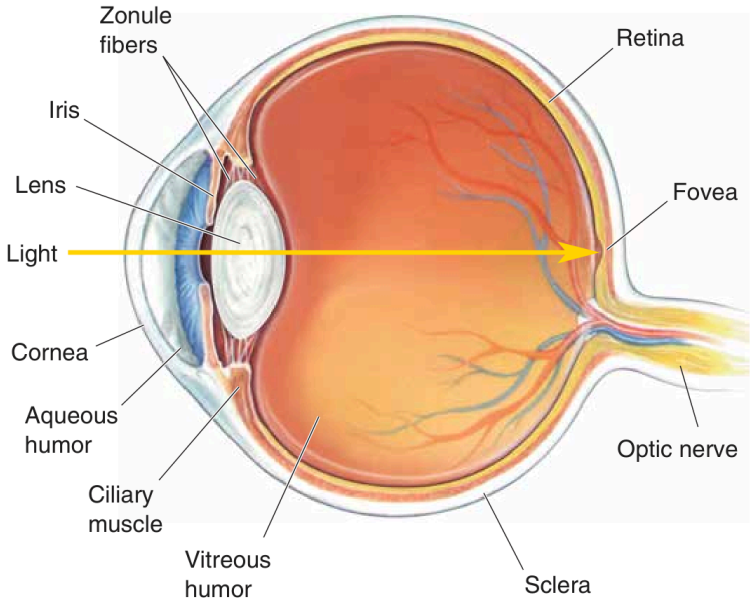


Figure 1.7: The eye in cross section. Reprinted from Bear et al. (2016, p. 299).

The neural pathway leaving the eye is called the *retinofugal projection* and begins with the *optic nerve* (Bear et al., 2016, pp. 333–334). After leaving the eye, the information passes through the optic chiasm—named after the Greek letter χ—where axon fibers originating from the nasal side of each retina cross from one axonal bundle to the other. This process is called *partial decussation* and splits the information into two bundles: one containing the left visual hemifield and one containing the right. The information then leaves the optic chiasm through the optic tract, as is visualized in Figure 1.8.

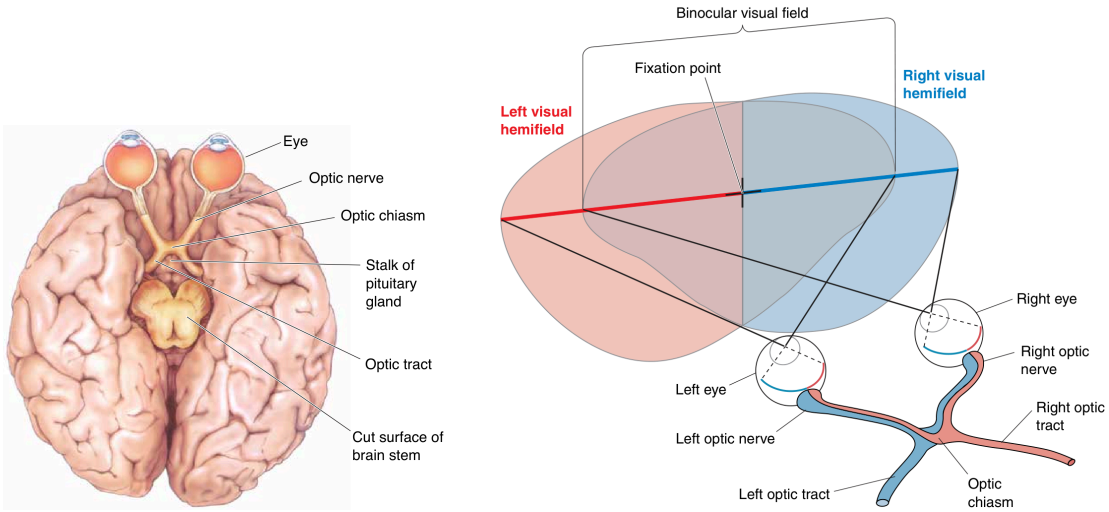


Figure 1.8: Retinofugal projection. The optic nerve, optic chiasm, and optic tract (left) and the information flow from the two visual hemifields (right). Reprinted from Bear et al. (2016, pp. 334–335).

The optic tract connects to the lateral geniculate nucleus (LGN)—named *geniculate* after a similar Latin word meaning "like a little knee" because of its cross-sectional bent-pancake-like structure (Bear et al., 2016, p. 338).

As illustrated in Figure 1.9, neurons in the LGN project directly to the visual cortex, located at the back of the brain (Bear et al., 2016, pp. 341–342). It is the part of the cerebral cortex—derived from Latin *cerebrum*, meaning brain, and *cortex*, meaning bark—that is responsible for most of the visual processing.

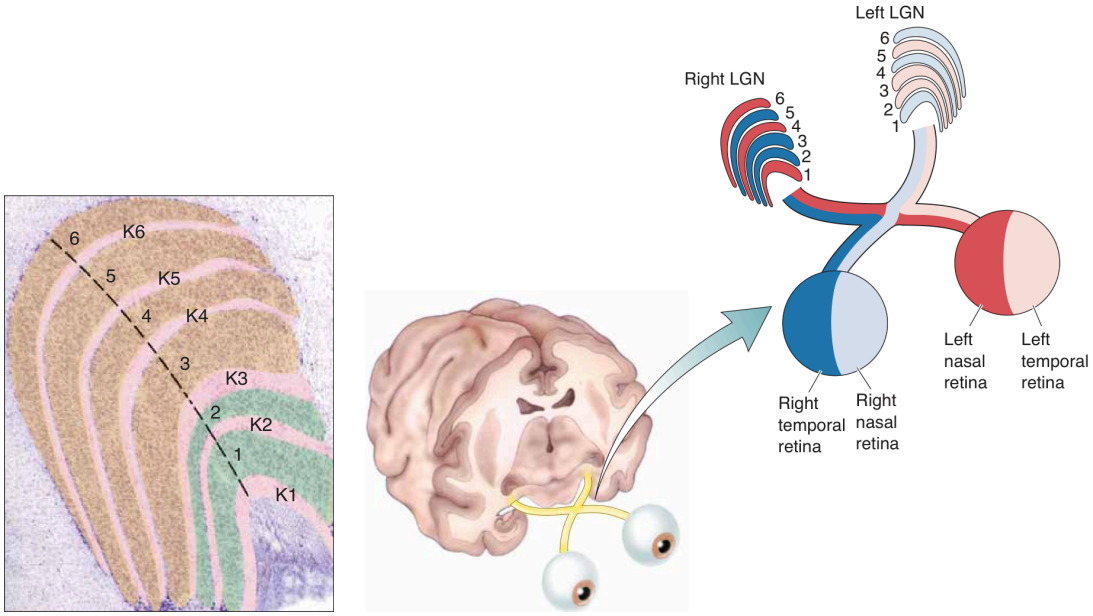


Figure 1.9: LGN organization. The layers of LGN (left) and the mapping of retinal inputs to LGN layers (right). Adapted from Bear et al. (2016, pp. 339–340).

The first area of the visual cortex is the primary visual cortex—also known as V1, striate cortex, or area 17—which receives the majority of the LGN's output (Bear et al., 2016, pp. 341–342, 356–358). The remaining areas are known as *extrastriate visual areas* and include V2, V3, V4, V5/MT, and IT, and are illustrated in Figure 1.10.

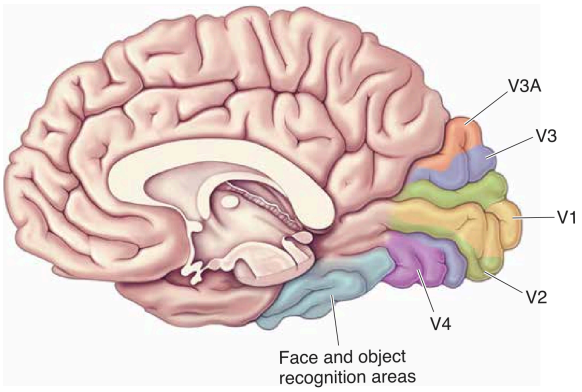


Figure 1.10: Visual areas in the human brain. Adapted from Bear et al. (2016, p. 357).

1.2.2 Retinotopy

An important organizational feature of the visual system is *retinotopy* (Bear et al., 2016, pp. 342–343). In this organization, neighboring neurons in the retina project to neighboring neurons in the LGN and, later, in the primary visual cortex.

Retinotopy is not 1:1. For example fovea, which is the part of the retina most densely populated with photoreceptors, sees only about 1–2 ° of the visual field (Coubard et al., 2014). As a result, the visual angle of the fovea becomes overrepresented and is magnified in the retinotopical mapping.

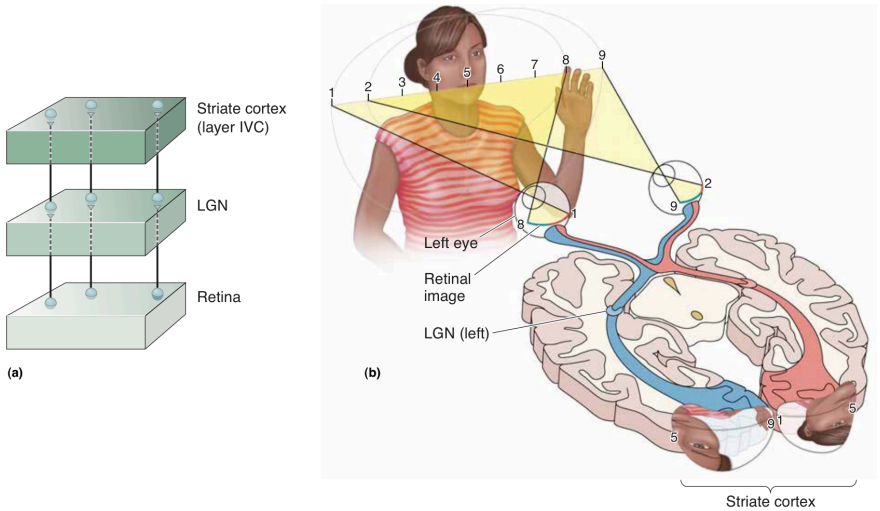


Figure 1.11: The retinotopic map in the striate cortex. **(a)** Neighboring locations on the retina project to neighboring locations in the LGN and striate cortex (V1). **(b)** The visual space represented in the V1. Reprinted from Bear et al. (2016, p. 343).

1.2.3 Retina

As explained by Bear et al. (2016, pp. 304–306) and illustrated in Figure 1.12, the neurons in the retina are organized into 5 classes, of which 3 classes (photoreceptors, bipolar cells, and ganglion cells) form the feed-forward pathway, and the remaining 2 classes (horizontal cells and amacrine cells) provide lateral connections.

All retinal neurons, with the exception of ganglion cells, do not fire action potentials. Instead, they release neurotransmitter in proportion to their membrane potential (so called *graded potentials*) (Bear et al., 2016, p. 312, 319). Photoreceptors (rods and cones) additionally also react to light, or more accurately, darkness. When a shadow comes across a photoreceptor, the photoreceptor depolarizes, increasing its neurotransmitter release.

The response properties of retinal and other visual neurons are commonly described using *receptive fields* (Bear et al., 2016, pp. 320–323). A neuron's receptive field is the region of the retina where a change in stimulus results in changes to that neuron's activity. It is typically drawn as the pattern that most effectively activates that neuron. For example, the receptive field of a single photoreceptor is a small black spot corresponding to that photoreceptor's exact location on the retina.

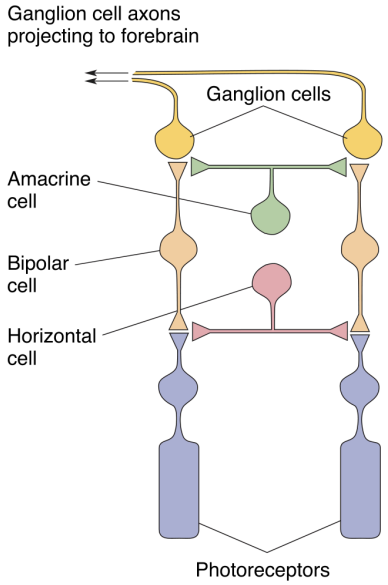


Figure 1.12: The basic system of retinal information processing. Reprinted from Bear et al. (2016, p. 305).

As visualized in Figure 1.13, bipolar cells receive direct input from photoreceptors in the *receptive field center*, as well as indirect input—relayed through the horizontal cell—from the neurons in the *receptive field surround*. The bipolar cell reacts *antagonistically* to the direct and indirect input, meaning that it either prefers darkness in the center and light in the surround, or light in the center and darkness in the surround.

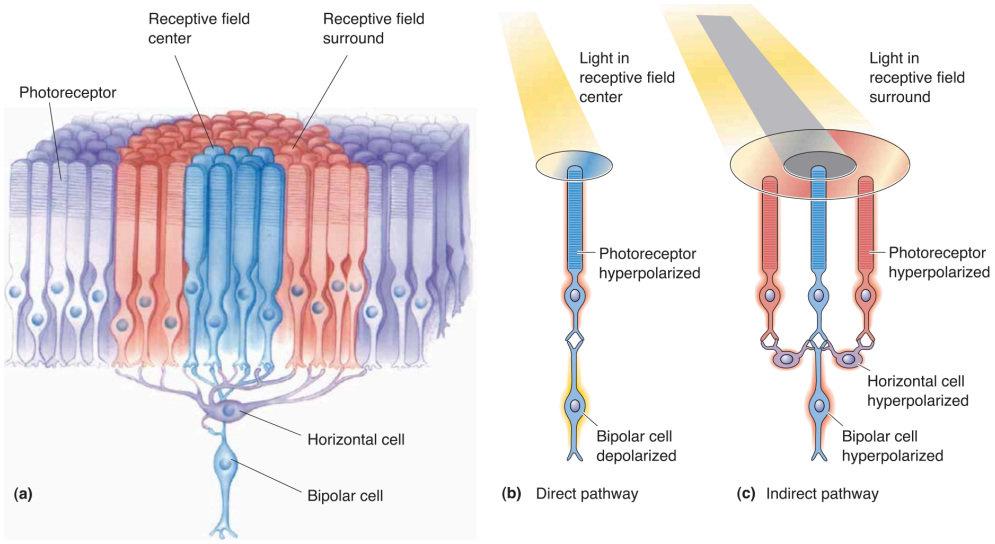


Figure 1.13: Direct and indirect pathways from photoreceptors to bipolar cells. **(a)** Bipolar cells receive direct central input and indirect surround input via photoreceptors and horizontal cells. **(b)** An ON-center bipolar cell is depolarized by light in the receptive field center via the direct pathway. **(c)** Light in the receptive field surround hyperpolarizes the ON-center bipolar cell via horizontal cells. Reprinted from Bear et al. (2016, p. 322).

This architecture results in two parallel channels: ON-center cells (preferring light in the center), and OFF-center cells (preferring darkness in the center) (Bear et al., 2016, pp. 321–325, 332). In general, these cells are invariant to the absolute brightness; they only react to the difference in light intensity between the center and the surround.

Another class of laterally connecting neurons are the amacrine cells, which come in a variety of types with different purposes (Masland, 2012). For example, one type of amacrine cell adds additional directional sensitivity to the ON-center/ OFF-center receptive fields.

For modelling the retinal output, the most important neurons are the ganglion cells, which serve as the output of the retina (Bear et al., 2016, pp. 324–325). These cells share essentially the same antagonistic center-surround receptive fields as bipolar cells, as illustrated by the activation patterns in Figure 1.14.

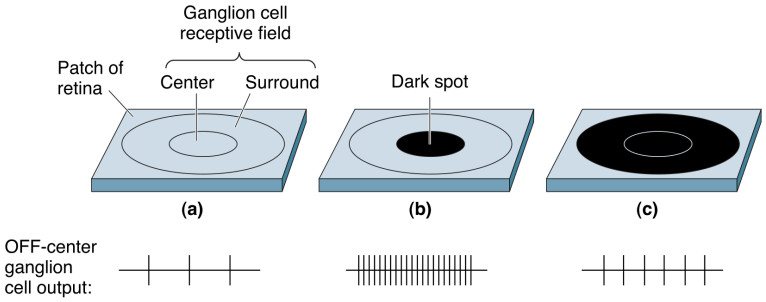


Figure 1.14: A center-surround ganglion cell receptive field. **(a, b)** An OFF-center ganglion cell responds with a barrage of action potentials to a dark spot in its receptive field center. **(c)** If the dark spot includes the receptive field surround, the response is greatly reduced. Reprinted from Bear et al. (2016, p. 324).

1.2.4 Lateral Geniculate Nucleus

The lateral geniculate nucleus (LGN) receives input from both retinas and organizes these inputs into distinct layers while preserving retinotopical organization (Bear et al., 2016, pp. 339–340). This lamination retains the separation of visual information based on which eye it originated from. In addition, the lamination also separates magnocellular cells, specialized for high temporal resolution (movement), from parvocellular cells, specialized for high spatial resolution (detail/color). This lamination is shown in Figure 1.15.

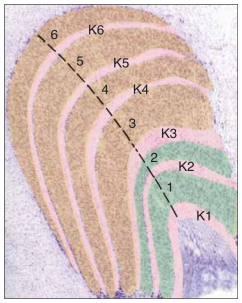


Figure 1.15: The organization of the LGN. Magnocellular cells are shown in green and parvocellular in orange. Bear et al. (2016, p. 340).

Interestingly, the LGN receives 80% of its inputs from the visual cortex, yet it maintains the same center-surround receptive field properties from the retinal outputs (Bear et al., 2016, p. 341).

While LGN is often thought of as a relay station, the extensive feedback from visual cortex suggests a more significant role. Since the LGN regulates how visual cortex gains access to visual information, its role likely includes a top-down attention mechanism modulating the outgoing information (Derrington, 2001).

1.2.5 Primary Visual Cortex (V1)

The primary destination of neurons from the LGN is the primary visual cortex (V1), which is the main biological reference point for this thesis (Bear et al., 2016, p. 341).

Like all cortical areas, V1 is organized into horizontal layers (Bear et al., 2016, pp. 343–344). To maintain consistency with the standard six-layer convention of the neocortex (I–VI), V1 is subdivided into the nine layers shown in Figure 1.16: I, II, III, IVA, IVB, IVCα, IVCβ, V, and VI. Feed-forward input from the LGN enters primarily layer IV and is then passed to superficial layers II and III (Bear et al., 2016, pp. 344–345). Although V1 contains additional recurrent, lateral, and feedback connections, this direct feed-forward route is relatively shallow compared with the depth of modern convolutional networks.

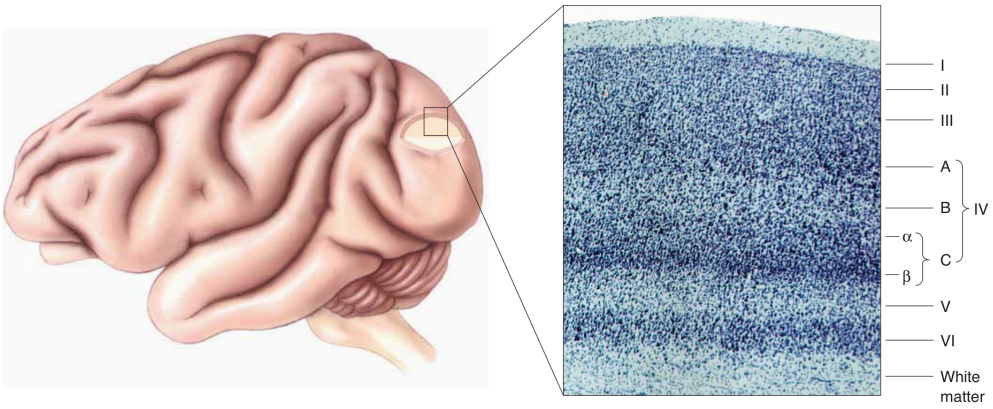


Figure 1.16: The lamination of the striate cortex. The purple dots represent neuron cell bodies. Reprinted from Bear et al. (2016, p. 344).

The visual system processes information via three partially parallel streams: the *magnocellular* (motion), *parvocellular* (fine shape/detail), and *koniocellular* (color) pathways (Bear et al., 2016, pp. 254–255). This thesis does not attempt to model these pathways separately. Instead, we focus on the aspects of V1 most relevant to the proposed architecture: local receptive fields and spatially structured connectivity.

As in most cortical areas, connectivity within V1 is primarily radial: neurons tend to connect perpendicular to the cortical surface, especially within the same vertical column (Bear et al., 2016, p. 345). This organization causes neurons along a vertical axis to share features such as ocular dominance or retinotopic position. Some neurons in layers II and III also form lateral connections and send axon collaterals to extrastriate areas such as V2 and V3 (Bear et al., 2016, p. 345).

1.2.6 Orientation Selectivity and Maps in V1

Most neurons in V1 have receptive fields corresponding to a particular orientation of a bar of light, as illustrated in Figure 1.17 (Bear et al., 2016, pp. 348–349). Although these cells can be described informally as edge detectors, Daugman (1985) proposed that their receptive fields are well modeled mathematically by Gabor functions. This theoretical framework was empirically verified by Jones & Palmer (1987), who demonstrated that the receptive fields of simple cells in cat striate cortex match the structure of a Gabor filter.

A Gabor filter, named after Denis Gabor, is a function obtained by modulating a sinusoidal plane wave with a Gaussian envelope (Daugman, 1985). Figure 1.18 shows examples of filters obtained by varying the orientation and phase parameters of the Gabor function.

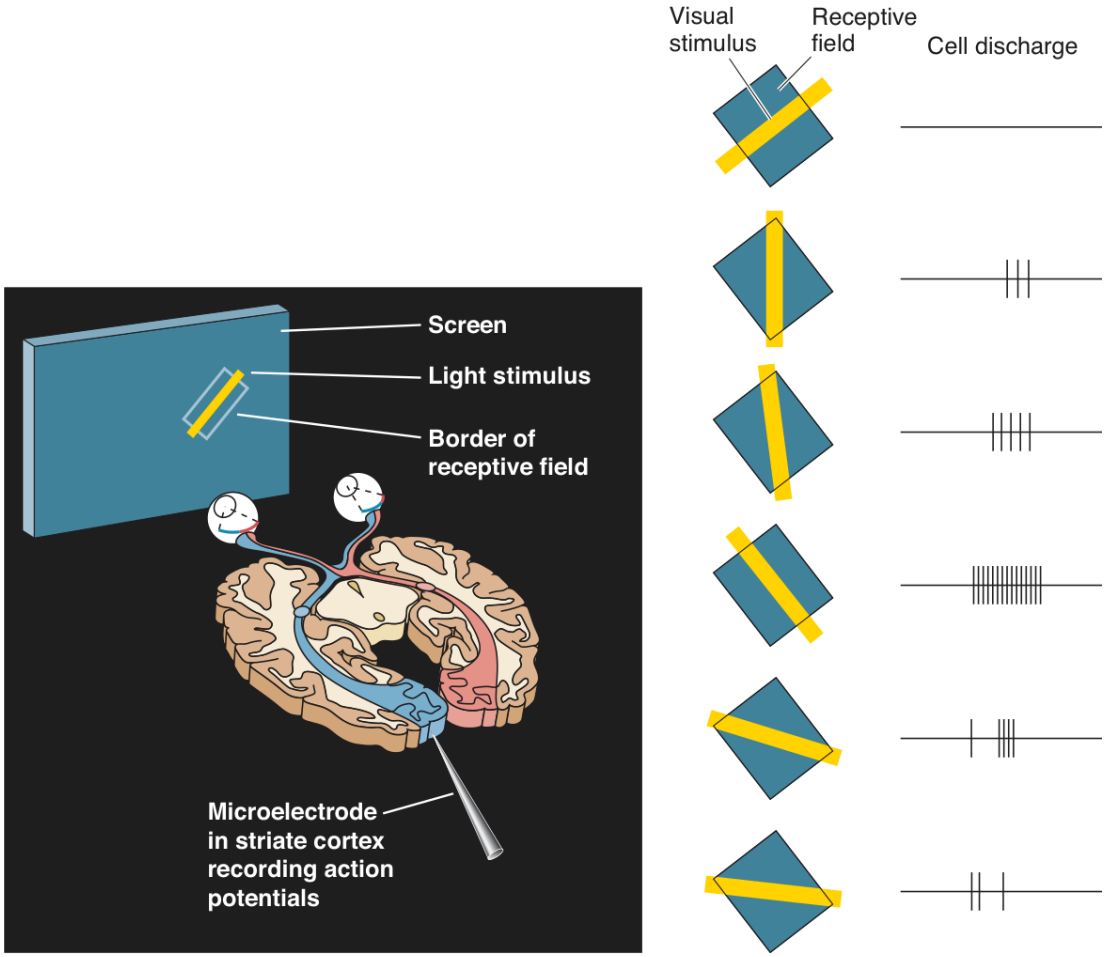


Figure 1.17: Orientation selectivity. Left: The responses of an orientation-selective neuron are monitored as visual stimuli are presented in its receptive field. Right: Light bars of various orientations elicit different spike firing rates. Adapted from Bear et al. (2016, p. 349).

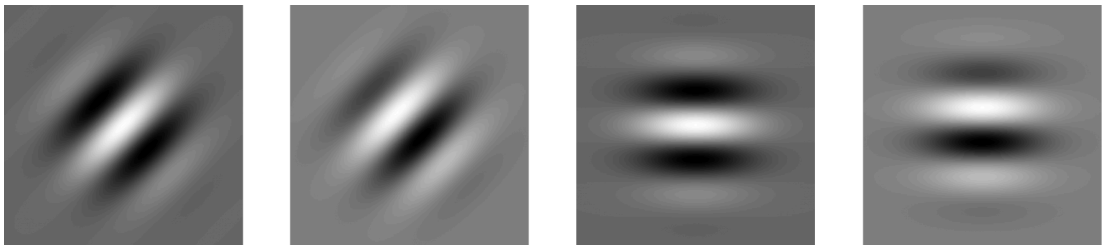


Figure 1.18: Gabor filter examples illustrating different angles and phases.

Neurons of V1 with receptive fields that can be mapped directly onto the retina are known as simple cells (Bear et al., 2016, pp. 351–353). In contrast, complex cells have receptive fields that are harder to visualize; for example, they can act as a functional 'OR' operator between multiple simple cells with the same orientation, creating shift-invariant edge detectors (Figure 1.19).

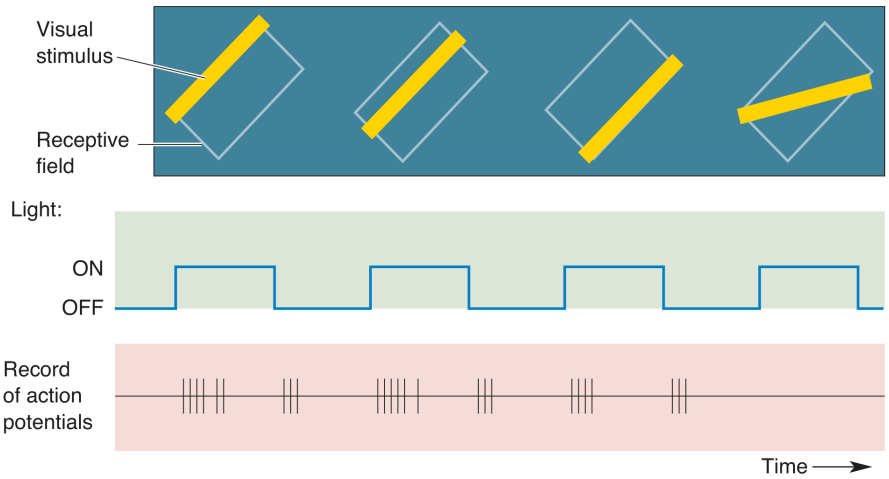


Figure 1.19: A complex cell receptive field. Like a simple cell, a complex cell responds best to a bar of light at a particular orientation. However, responses occur regardless of position in the receptive field. Bear et al. (2016, p. 353).

Crucially, the orientation preference of the neurons in V1 is not randomly distributed (Bear et al., 2016, pp. 348–350). Rather, neurons are organized into a smooth cortical map, with neurons with similar orientation preference clustered together. One such orientation preference map can be seen in Figure 1.20.

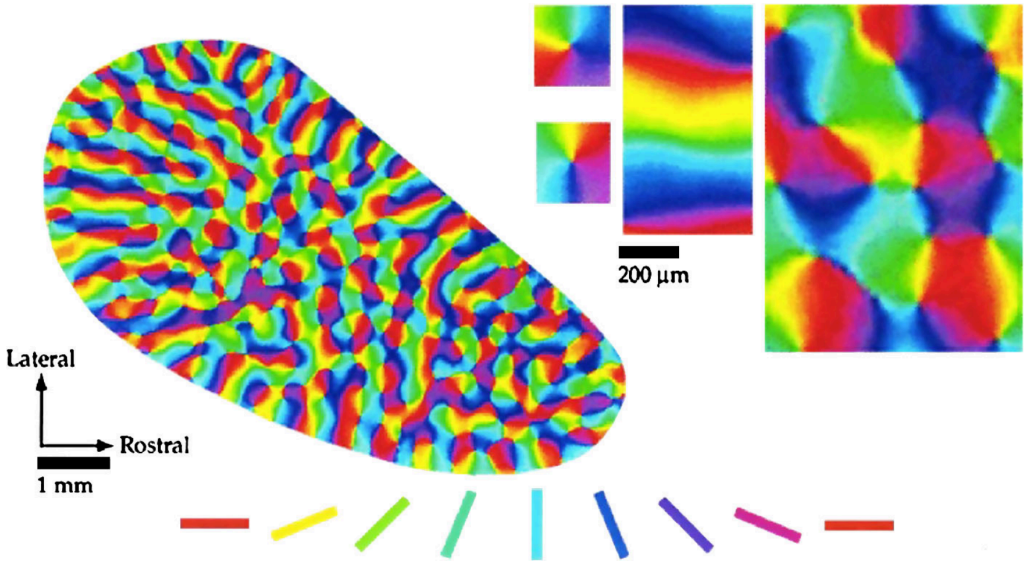


Figure 1.20: An orientation preference map observed in the visual cortex of a tree shrew. Singular pinwheels are detailed in the upper right corner. Reprinted from Afgoustidis (2015, p. 2), originally from Bosking et al. (1997, p. 2115).

Orientation preference maps often give rise to many so-called *pinwheels*: structures in which neuronal orientation preferences rotate continuously through all 180° around a central singularity (Ohki et al., 2006). Example of these pinwheels can be seen in Figure 1.20. The density of these pinwheels scales inversely with the *hypercolumn size*—the cortical area required to represent a full range of orientation preferences for both ocular dominance pathways (Kaschube et al., 2010). For example, in macaque's primary visual cortex, a hypercolumn spans approximately $2 \times 2\,\text{mm}$.

1.2.7 Extrastriate Cortex

Further visual processing beyond striate cortex (V1) occurs in the extrastriate cortex in two main information pathways: *dorsal* (positioned towards the top of the brain) and *ventral* (positioned towards the bottom of the brain) (Bear et al., 2016, p. 356). As illustrated in Figure 1.21, these streams do not follow a strict serial hierarchy. Nevertheless, Bear et al. (2016, p. 358) note that there appears to be a progression of areas across which the representational complexity increases.

- The dorsal stream, which includes areas V2, V3, MT, and MST, is primarily involved in motion processing (Bear et al., 2016, p. 356).
- The ventral stream, on the other hand, is thought to be involved in object recognition and general perception of the visual world (Bear et al., 2016, p. 356). It includes areas V2, V3, V4, and IT.

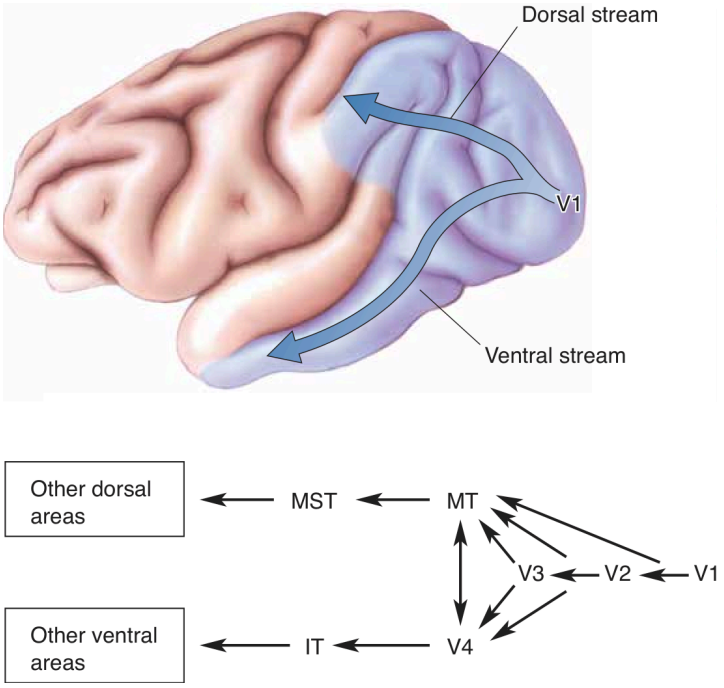


Figure 1.21: Extrastriate cortex in the macaque monkey brain. Top: Dorsal and ventral processing streams. Bottom: The flow of information in the dorsal and ventral streams. Bear et al. (2016, p. 357).

This increased representational complexity can be seen, for example, in V2, where neurons respond to higher-order statistical dependencies found in natural textures (Freeman et al., 2013). At the furthest extent of the ventral stream, this culminates in area IT, which is responsible for the detection of complex objects, such as faces (Bear et al., 2016, pp. 360–362).

2 Deep Learning Background

2.1 Artificial Neuron

While more biologically accurate models of a neuron exist, such as the conductance-based Hodgkin-Huxley model (Hodgkin & Huxley, 1952), or the more computationally efficient Leaky Integrate-and-Fire (LIF) model (Lapicque, 1907), modern deep learning predominantly uses the artificial neuron originally introduced by McCulloch & Pitts (1943) (see Figure 2.1), which serves as the fundamental building block for contemporary architectures (Goodfellow et al., 2016, pp. 12–18).

A key distinction between these models is that while the Hodgkin-Huxley and leaky integrate-and-fire models operate by producing discrete action potentials (spikes) similar to biological neurons, the artificial neuron model is *rate-coded*, meaning that the neuron's output represents the neuron's average spiking frequency given its current inputs (Dayan & Abbott, 2001, pp. 8–14, 231–239).

Formally, the output y of an artificial neuron is calculated by taking the weighted sum of its inputs $\boldsymbol{x} = (x_1, ..., x_N)$, adding the bias b, and applying the activation function f:

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right) = f(\boldsymbol{w}^\top \boldsymbol{x} + b)\,,$$

where $\boldsymbol{w} = (w_1, ..., w_N)$ is the vector of weights.

While the original artificial neuron by McCulloch & Pitts (1943) utilized a hard-threshold activation function:

$$f(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ if } z < 0\,, \end{cases}$$

modern deep learning typically employs continuous activation functions, such as the Rectified Linear Unit (ReLU):

$$\text{ReLU}(z) = \begin{cases} z \text{ if } z \geq 0 \\ 0 \text{ if } z < 0\,, \end{cases}$$

or the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}\,.$$

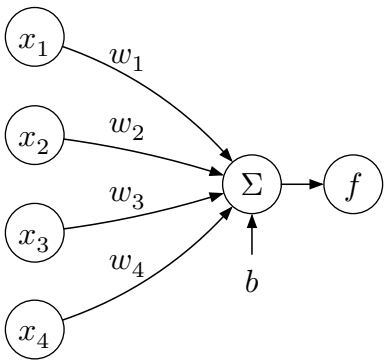


Figure 2.1: Artificial neuron model.

2.2 Feedforward Neural Networks

By arranging these artificial neurons into a sequence of three layers—input layer, *hidden layer*, and output layer—we construct a basic feedforward neural network (Figure 2.2). According to the Universal Approximation Theorem (Leshno et al., 1993), such network can approximate any continuous function to an arbitrary precision, provided that the network is sufficiently wide, and that the selected activation function is non-polynomial.

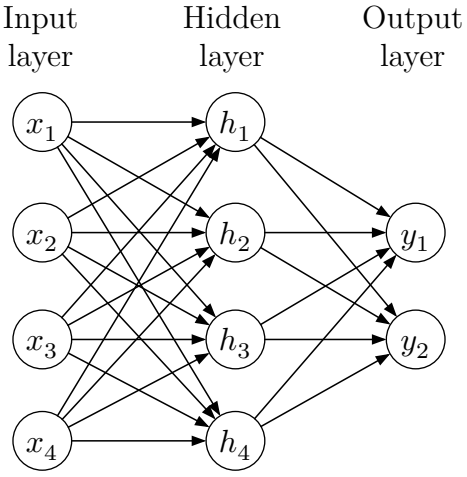


Figure 2.2: Fully-connected feedforward neural network with one hidden layer and two outputs.

However, feedforward neural networks with just one hidden layer are often computationally inefficient, as the number of neurons needed in the hidden layer can grow exponentially with the number of input features (Goodfellow et al., 2016, pp. 198–199). Furthermore, there exist functions that can be efficiently expressed using feedforward neural networks of some depth d, but require significantly larger networks if constrained to fewer than d layers (Eldan & Shamir, 2016; Goodfellow et al., 2016, pp. 198–199). This depth then enables the network to learn a feature hierarchy, with each layer utilizing the representations learned by the previous layer, similar to, for example, the ventral stream in the human brain (Felleman & Van Essen, 1991).

To construct these deep architectures, neurons are typically organized into fully-connected layers. In this configuration, every neuron in a given layer receives input from every neuron in the preceding layer. Formally, we can vectorize the individual neuron operations into a single layer-wise transformation. The output of layer l, denoted as $\boldsymbol{h}^{(l)} \in \mathbb{R}^m$, is computed as:

$$\boldsymbol{h}^{(l)} = f^{(l)}\big(\boldsymbol{h}^{(l-1)}\boldsymbol{W}^{(l)} + \boldsymbol{b}^{(l)}\big)\,,$$

where $\boldsymbol{W}^{(l)} \in \mathbb{R}^{m\times n}$ is the weight matrix and $\boldsymbol{b}^{(l)} \in \mathbb{R}^n$ is the bias, with $\boldsymbol{W}^{(l)}_{i,j}$ representing the strength of connection between the i-th neuron of the previous layer and j-th neuron of the current layer.

2.2.1 Probabilistic Interpretation

We interpret the output of a neural network as a probability distribution over possible targets. In classification tasks, for example, the final layer uses the *softmax* activation function, which transforms the *logits* $\boldsymbol{z}$ into a categorical distribution $\hat{\boldsymbol{y}}$:

$$\hat{y}_k = \text{softmax}(\boldsymbol{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{n} e^{z_j}} ,$$

where $k \in \{1, ..., K\}$ is the class index, K is the total number of classes, and $\hat{y}_i$ can be interpreted as the probability that the current example $\boldsymbol{x}$ belongs to class i (Goodfellow et al., 2016, pp. 184–187).

For a single input example $\boldsymbol{x}$, the entire classification network can then be viewed as a conditional probability model (Goodfellow et al., 2016, p. 133):

$$P(y = i \mid \boldsymbol{x}; \theta) = \hat{y}_i ,$$

where θ denotes a set of all *parameters* (weights and biases) in the network.

2.3 Training Neural Networks

To train a neural network, we consider a dataset of input-target pairs:

$$\mathcal{D} = \left\{ \left(\boldsymbol{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N} ,$$

where each pair was sampled independently from the same *data-generating distribution* (i.i.d) (Goodfellow et al., 2016, pp. 110–111).

The objective of training is to find the parameters θ that minimize the discrepancy between the network's predictions and the ground-truth targets, particularly on new, previously unseen inputs drawn from the same data-generating distribution (Goodfellow et al., 2016, p. 110).

2.3.1 Maximum Likelihood Estimation

A principled approach for finding such parameters is using the *maximum likelihood principle*, which selects the parameters $\hat{\theta}$ that maximize the likelihood of the entire dataset under the model P:

$$\begin{aligned}
\hat{\theta} &= \arg\max_{\theta} \prod_{i=1}^{N} P\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \theta\right) \\
&= \arg\max_{\theta} \log \left(\prod_{i=1}^{N} P\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \theta\right) \right) \\
&= \arg\max_{\theta} \sum_{i=1}^{N} \log P\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \theta\right) \\
&= \arg\min_{\theta} \sum_{i=1}^{N} - \log P\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \theta\right) ,
\end{aligned}$$

where we utilize the fact that the logarithm is a monotonically increasing function to transform the product into a sum (Goodfellow et al., 2016, pp. 131–133).

The resulting value $\hat{\theta}$ is known as the *maximum likelihood estimate* (MLE). Finding this estimate is equivalent to minimizing the scalar loss $\mathcal{L}$:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log P\big(y^{(i)} \mid \boldsymbol{x}^{(i)}; \theta\big)\,,$$

which is referred to either the *negative log-likelihood loss* ($\mathcal{L}_{\text{NLL}}$), or due to its form matching the equation for *conditional cross-entropy*, the *categorical cross-entropy loss* ($\mathcal{L}_{\text{CE}}$).

2.3.2 Minibatch SGD

Training of neural networks is typically performed via the *stochastic gradient descent* algorithm (SGD) (Goodfellow et al., 2016, pp. 151–153).

The training algorithm, as detailed in Listing 2.1, splits the dataset into batches of fixed size, and for each batch, estimates the gradient of the chosen loss function ℓ with respect to its parameters θ, which are then updated accordingly (Goodfellow et al., 2016, p. 294). This process is repeated across multiple epochs until the model converges.

inputs: neural network N,
dataset $\mathcal{D}$ partitioned into batches $\mathcal{B}$,
loss function ℓ,
learning rate η.
for each batch $(\boldsymbol{X}, \boldsymbol{y}) \in \mathcal{B}$:
compute predictions $\hat{\boldsymbol{y}} \leftarrow N(\boldsymbol{X})$
compute the scalar loss $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \ell(\hat{y}_i, y_i)$
compute the gradient estimate $\hat{g}_\theta \leftarrow \nabla_\theta \mathcal{L}$
perform one update step $\theta \leftarrow \theta - \eta \hat{g}_\theta$

Listing 2.1: Stochastic gradient descent algorithm (SGD) for optimizing a neural network using mini-batches of data.

Several variants of SGD have been proposed to improve its convergence properties. Most notably Adam (Kingma & Ba, 2017), which incorporates adaptive learning rates for each parameters, and can significantly speed up the training in complex neural network architectures.

2.3.3 Gradient Computation

In modern deep learning frameworks such as PyTorch (Paszke et al., 2017), TensorFlow (Martín~Abadi et al., 2015), and JAX (Bradbury et al., 2018), the computation of a neural network is represented as a Directed Acyclic Graph (DAG) of mathematical operations performed on tensors (Goodfellow et al., 2016, p. 168).

During the *forward pass*, the network computes values following a topological ordering of the DAG. During each operation, the underlying *automatic differentiation* (autograd) engine records the operation it performed and all necessary intermediate tensors required to calculate the gradient (Paszke et al., 2017).

The calculation of gradients is performed during the *backward pass*, using the *backpropagation algorithm*, which traverses the DAG in reverse topological order, and utilizes the chain rule to calculate the gradients of all intermediate tensors and parameters with respect to the defined loss $\mathcal{L}$ (Goodfellow et al., 2016, pp. 204–207).

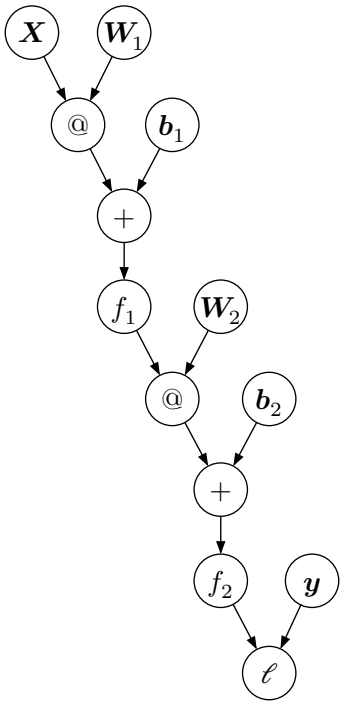


Figure 2.3: A computational graph for the neural network defined in Figure 2.2. Each leaf node is either a parameter ($\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2$) or a fixed tensor (input $\boldsymbol{X}$, target $\boldsymbol{y}$). Each non-leaf node represents an operation performed on the nodes leading into it, with @ denoting matrix multiplication and ℓ the loss function.

Given a tensor Y, given by $\mathsf{Y} = f(\mathsf{X}_1, ..., \mathsf{X}_n)$, and its gradient $g_{\mathsf{Y}} = \frac{\partial \mathcal{L}}{\partial \mathsf{Y}}$, the gradient of each argument X_i is computed via the chain rule:

$$g_{\mathsf{X}_i} = \frac{\partial \mathcal{L}}{\partial \mathsf{X}_i} = \frac{\partial \mathcal{L}}{\partial \mathsf{Y}} \cdot \frac{\partial \mathsf{Y}}{\partial \mathsf{X}_i} = g_{\mathsf{Y}} \cdot \frac{\partial f(\mathsf{X}_1, ..., \mathsf{X}_n)}{\partial \mathsf{X}_i} \,.$$

In cases where the tensor X_i was used to calculate multiple tensors $(\mathsf{Y}_1, ..., \mathsf{Y}_m)$, the gradient is given by:

$$g_{\mathsf{X}_i} = \frac{\partial \mathcal{L}}{\partial \mathsf{X}_i} = \sum_{j=1}^{m} \frac{\partial \mathcal{L}}{\partial \mathsf{Y}_j} \cdot \frac{\partial \mathsf{Y}_j}{\partial \mathsf{X}_i} = \sum_{j=1}^{m} \left(g_{\mathsf{Y}_j} \cdot \frac{\partial f_j(\mathsf{X}_i, ...)}{\partial \mathsf{X}_i} \right) ,$$

where f_j is the operation used to calculate tensor Y_j.

The backpropagation process is initialized at the scalar loss node, where the gradient of loss with respect to itself is given by:

$$g_{\mathcal{L}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1 \,.$$

2.4 Computer Vision

While computer vision (CV) was originally inspired by the mammalian visual cortex (DiCarlo et al., 2012; Yamins & DiCarlo, 2016), modern architectures are increasingly prioritizing task performance over biological plausibility (Hassabis et al., 2017).

Currently, the field is dominated by two paradigms: Convolutional Neural Networks (CNNs) (Goodfellow et al., 2016, p. 330), which rely on local connectivity and shared kernels, and Vision Transformers (ViTs) (Dosovitskiy et al., 2020), which utilize global self-attention mechanisms. This thesis focuses on the former, as the local connectivity of convolutions is closer to the biological structure of the primary visual cortex.

2.4.1 2D Convolutional Layer

Whilst it is typically referred to as convolution, the fundamental operation of a convolutional layer is cross-correlation, denoted by the symbol $\star$ (Goodfellow et al., 2016, p. 333).

Assuming a square kernel of size K, the kernel is a four-dimensional tensor $\mathsf{K} \in \mathbb{R}^{K \times K \times C_{\text{in}} \times C_{\text{out}}}$. The value of the output at the spatial position (i, j) and the channel o is given by:

$$(K \star I)_{i,j,o} = b_o + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} \sum_{c=0}^{C_{\text{in}}-1} \mathsf{I}_{i+m,j+n,c} \cdot \mathsf{K}_{o,c,m,n} \,,$$

where b_o is the bias term for output channel o (Goodfellow et al., 2016, pp. 331–334). Without padding, this transforms the input $\mathsf{I} \in \mathbb{R}^{H \times W \times C_{\text{in}}}$ into an output of size $\mathbb{R}^{(H-K+1) \times (W-K+1) \times C_{\text{out}}}$.

With stride s, the 2D convolution becomes:

$$(K \star I)_{i,j,o} = b_o + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} \sum_{c=0}^{C_{\text{in}}-1} \mathsf{I}_{si+m,sj+n,c} \cdot \mathsf{K}_{o,c,m,n} \,,$$

which results in further downscaling of the output dimensions.

The first fundamental principle of convolutions is *local connectivity*: each output unit is connected only to a small, spatially contiguous region of the input. This design, pioneered by Fukushima (1980), directly mimics the local receptive fields discovered in the mammalian primary visual cortex, where individual neurons respond only to stimuli within a physically constrained portion of the visual field, such as edges of specific orientations (Hubel & Wiesel, 1962).

The second fundamental principle is *weight sharing*: the same kernel is applied across all spatial locations of the input. While this enforces *translation invariance*—ensuring the same features are detected regardless the positioning of the object within the image—and provides high computational efficiency on modern hardware (LeCun et al., 1989), it represents a significant departure from biological architectures. Biological brains have no known mechanism that would allow them to share synaptic weights across different cortical regions (Pogodin et al., 2022).

Furthermore, convolutional neural networks rely on storing features in discrete channels, representing the same feature at each spatial point. This differs fundamentally from the biological brain, which tends to spread features out spatially across the cortical surface in smooth topographical maps (Bear et al., 2016, pp. 348–350). This idea is illustrated in Figure 2.4.

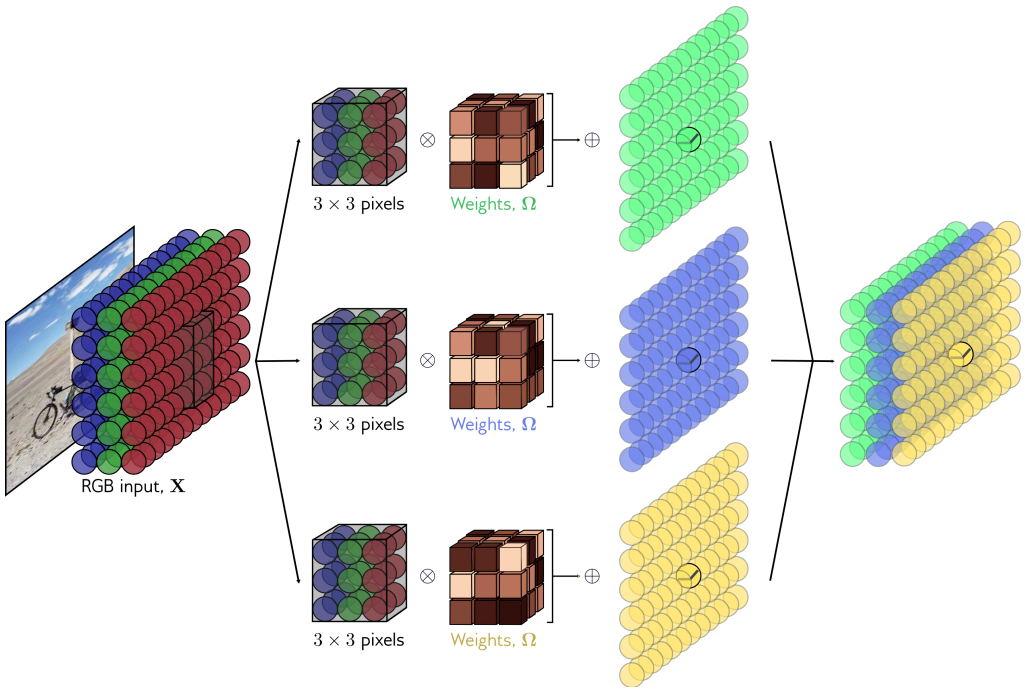


Figure 2.4: A 2D convolutional layer utilizing multiple different kernels to extract features and stack them onto separate output channels. Adapted from Straka (2026, slide 12), originally from Prince (2023, p. 172).

2.4.2 Convolutional Architectures

Convolutional neural networks came into the limelight with AlexNet (Krizhevsky et al., 2017), which demonstrated that deep learning architectures could significantly outperform traditional computer vision models. Following this breakthrough, several influential architectures emerged, such as VGG (Simonyan & Zisserman, 2015), Inception (Szegedy et al., 2014), and ResNet (He et al., 2015a).

In this work, we focus on ResNet (He et al., 2015a) as the primary baseline for our comparisons. ResNet introduced skip connections, illustrated in Figure 2.5, as a mechanism to allow gradients to bypass layers, effectively mitigating the vanishing gradient problem and improving convergence in very deep networks. A standard ResNet-34 architecture is illustrated in Figure 2.6.

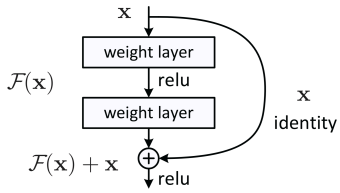


Figure 2.5: ResNet residual block. Reprinted from He et al. (2015a, p. 2).

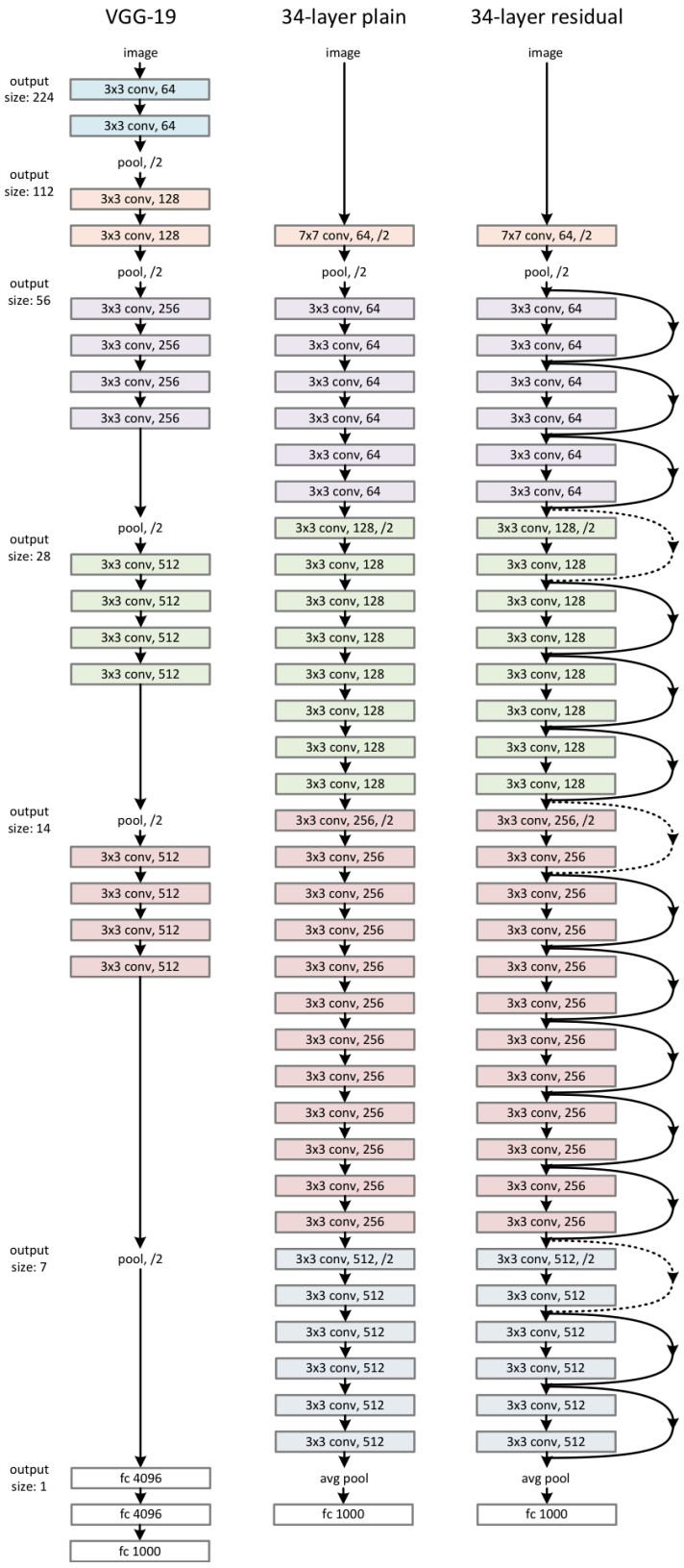


Figure 2.6: Example network architecture of ResNet-34. Left: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Reprinted from He et al. (2015a, p. 4).

Interestingly, while modern CNN architectures continue to increase depth to achieve better performance, the biological brain is surprisingly shallow in its feedforward pass. The primate ventral stream consists of roughly 5–10 stages before reaching high-level object representations (DiCarlo et al., 2012). However, unlike standard feedforward CNNs, the brain is characterized by massive recurrent connections. Notably, it has been shown that ResNet can be mathematically viewed as an unrolled recurrent neural network (Liao & Poggio, 2020). This suggests that the extreme depth of modern CNNs may effectively act as a proxy for the recurrent connections found in biological brains, implying that more biologically-inspired architectures might achieve similar performance by replacing depth with recurrent connections.

2.5 Hypernetworks

A hypernetwork is a neural network that generates the weights of another neural network (Ha et al., 2016). Instead of storing all weights of the target model directly as independent trainable parameters, a hypernetwork learns a separate weight-generating function. This allows the generated weights to depend on the input to the hypernetwork, making it possible to introduce structured dependencies between weights. Figure 2.7 illustrates an example hypernetwork that generates weights for different layers from layer indices and other information.

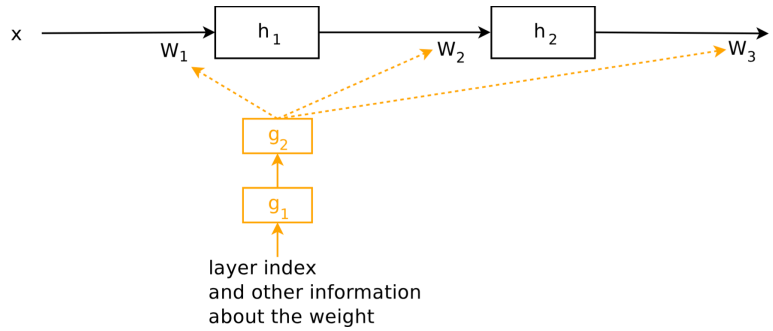


Figure 2.7: A hypernetwork generates the weights for a feedforward network. Black connections and parameters are associated the main network whereas orange connections and parameters are associated with the hypernetwork. Reprinted from Ha et al. (2016, p. 1).

2.6 Implicit Neural Representations

Implicit neural representations (INRs) use neural networks to represent continuous signals as functions over coordinates, rather than storing them only as discrete grids of values. For example, an image can be represented as a function mapping a two-dimensional coordinate (x, y) to a pixel intensity or color value. A particularly relevant type of INR is the sinusoidal representation network (SIREN), introduced by Sitzmann et al. (2020), which replaces standard activations such as ReLU with sinusoidal activations. This allows SIRENs to represent continuous signals with fine spatial detail, including high-frequency structure, while requiring a specialized initialization scheme for stable training.

3 Related Work

Several recent works have introduced topographic organization into deep neural networks through explicit auxiliary objectives. Topographic Deep Artificial Neural Networks (TDANNs) impose spatial constraints on artificial units by using a spatial loss that encourages nearby units to have more similar response profiles than distant units (Lee & DiCarlo, 2018). This shows that biologically inspired topographic organization can be encouraged in task-performing deep networks.

More recently, All-Topographic Neural Networks (All-TNNs) extended this direction toward end-to-end topographic models of visual cortex (Lu et al., 2025). These models develop smooth orientation maps in early layers and category-selective regions in later layers. However, they remain conceptually close to TDANN-style approaches in that topographic organization is encouraged through an explicit similarity-based objective.

A related but architecturally different approach was proposed by Poli et al. (2022). Instead of replacing convolutional architectures, this method keeps standard convolutional layers, assigns spatial positions to convolutional output channels, and introduces a topographic loss that encourages nearby channels to have similar activations. Unlike TDANN-style approaches and the model proposed in this thesis, topography is introduced within the existing channel structure of CNNs rather than through a more spatially explicit neural architecture.

The present thesis is closest in motivation to TDANN and All-TNN–style models, since it also aims to obtain cortical-like topographic organization in artificial visual networks. However, it differs in the mechanism used to obtain this organization. Rather than imposing topography through an auxiliary loss on neighboring units or channels, we modify the architecture itself: standard convolutional channels are replaced by two-dimensional neuronal sheets, and connectivity is localized, neuron-specific, and generated through shared hypernetwork-based parameterizations. Thus, topographic organization is not directly optimized as a similarity objective, but studied as an emergent consequence of the architectural constraints.

A further architectural distinction concerns the formulation of the local connectivity layer itself. The `LocalLinear` layer used in this thesis is a strict generalization of standard locally connected layers. Standard implementations of locally connected layers assume an integer-stride relationship between input and output grids, whereas `LocalLinear` explicitly defines the alignment between output-sheet positions and local regions of the input sheet. This allows mappings between neuronal sheets of arbitrary sizes and densities.

4 Methods

4.1 LocalLinear Layer

The core architectural element used in our thesis is the `LocalLinear` layer, which can be viewed as a two-dimensional convolutional layer without weight sharing. However, unlike standard implementations of this type of layer, typically referred to as `LocallyConnected2D`, our formulation allows mappings between neuronal sheets of arbitrary size and density.

A `LocalLinear` layer defines a mapping from an input sheet of shape $(H_{\text{in}}, W_{\text{in}}, C_{\text{in}})$ to an output sheet of shape $(H_{\text{out}}, W_{\text{out}}, C_{\text{out}})$:

$$\text{LL}_\theta : \mathbb{R}^{H_{\text{in}} \times W_{\text{in}} \times C_{\text{in}}} \to \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}} ,$$

where H, W, and C denote the height, width, and the number of channels, respectively.

To define this mapping, the layer requires a kernel K of shape $(H_{\text{out}}, W_{\text{out}}, C_{\text{out}}, C_{\text{in}}, k_{\text{h}}, k_{\text{w}})$ and a bias B of shape $(H_{\text{out}}, W_{\text{out}}, C_{\text{out}})$, where H_{out} and W_{out} are the output height and width, C_{out} and C_{in} are the number of output and input channels, and k_{h} and k_{w} correspond to the height and width of the local patch, analogous to the kernel size in standard convolutions.

This formulation is easiest to understand by comparing it to a standard two-dimensional convolution, which computes the output at sheet index (i, j) and channel o as

$$\text{Conv2D}(\mathsf{I}; \mathsf{K}, \boldsymbol{b})_{i,j,o} = b_o + \sum_{m=0}^{k_{\text{h}}-1} \sum_{n=0}^{k_{\text{w}}-1} \sum_{c=0}^{C_{\text{in}}-1} \mathsf{I}_{si+m, sj+n, c} \mathsf{K}_{o,c,m,n} ,$$

where $\mathsf{I} \in \mathbb{R}^{H_{\text{in}} \times W_{\text{in}} \times C_{\text{in}}}$, $\mathsf{K} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_{\text{h}} \times k_{\text{w}}}$ is the kernel, $\boldsymbol{b} \in \mathbb{R}^{C_{\text{out}}}$ is the bias, and s is the stride. To improve readability, output indices are shown in red, while input and kernel indices are shown in blue.

In contrast, our `LocalLinear` layer computes that same output unit as

$$\text{LL}(\mathsf{I}; \mathsf{K}, \mathsf{B})_{i,j,o} = \mathsf{B}_{i,j,o} + \sum_{m=0}^{k_{\text{h}}-1} \sum_{n=0}^{k_{\text{w}}-1} \sum_{c=0}^{C_{\text{in}}-1} \mathsf{I}_{\phi(i)+m, \phi'(j)+n, c} \mathsf{K}_{i,j,o,c,m,n} ,$$

where the kernel $\mathsf{K} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}} \times C_{\text{in}} \times k_{\text{h}} \times k_{\text{w}}}$ and bias $\mathsf{B} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}}$ are expanded to depend explicitly on the output location. Furthermore, our formulation replaces the stride with functions $\phi : \{0, ..., H_{\text{out}} - 1\} \to \{0, ..., H_{\text{in}} - k_{\text{h}}\}$ and $\phi' : \{0, ..., W_{\text{out}} - 1\} \to \{0, ..., W_{\text{in}} - k_{\text{w}}\}$, which map each output index to the corresponding input index at which the local kernel window begins.

This use of ϕ and ϕ' allows mappings between neuronal sheets of arbitrary size and density.

4.1.1 Implementation

Our implementation supports two configuration modes. In the first mode, the layer is specified through a physical description of the neuronal layers, consisting of their physical size, neuronal density, and the number of neuronal units[3], together with a connection radius r defined in the same physical units.

The kernel size is then chosen as the smallest kernel sufficient to cover arbitrarily positioned circle of radius r:

$$k_{\mathrm{h}} = k_{\mathrm{w}} = \left\lceil \frac{2r}{d} \right\rceil + 1$$

where d is the distance between neighboring neurons on the input sheet.

The functions ϕ and ϕ' are implemented as lookup tables. Their values are obtained using the procedure illustrated in Figure 4.1, which determines, for each output index i or j, the corresponding input index ($\varphi(i)$ or $\varphi'(j)$) at which the local kernel window begins.

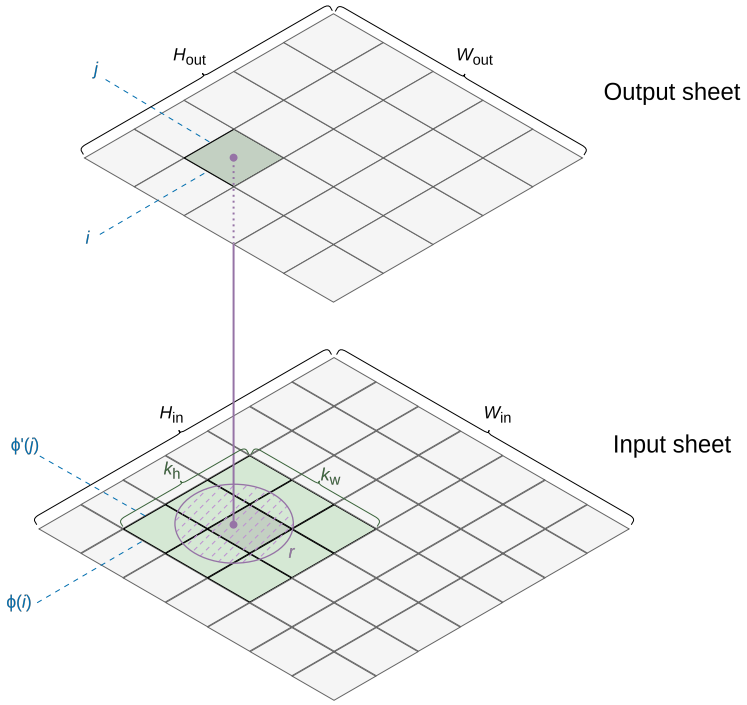


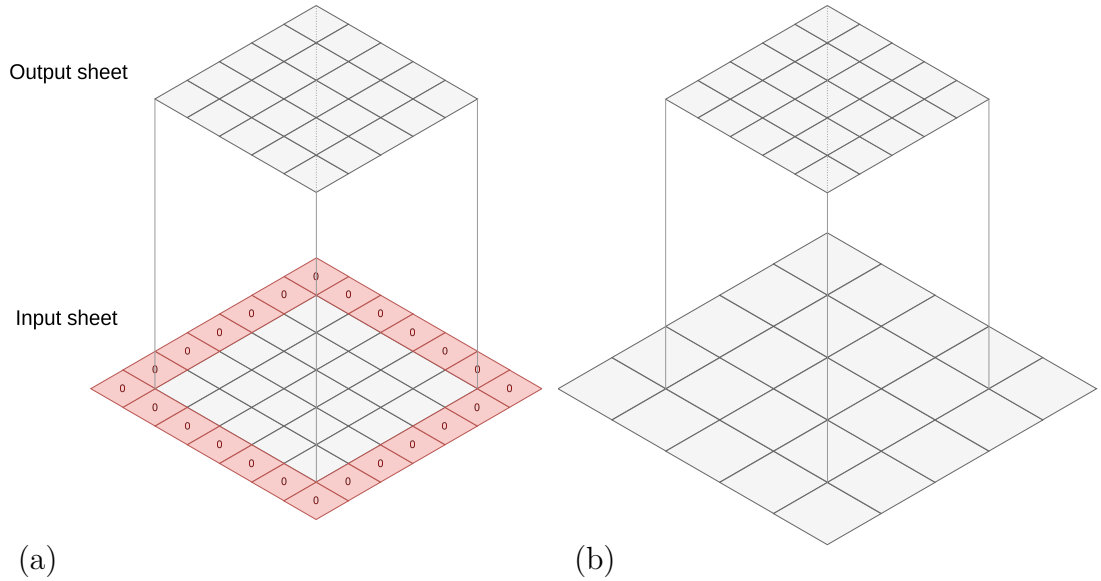
Figure 4.1: Finding $\phi(i)$ and $\phi'(j)$ in a `LocalLinear` layer. After finding the kernel size $(k_{\mathrm{h}}, k_{\mathrm{w}})$ required to support the selected connection radius r, the physical center of the neuron (i, j) is projected from the output neuronal sheet onto the input neuronal sheet, where the nearest neuron is selected as the center for the kernel window.The values, $(\phi(i), \phi'(j))$ are then set to the top-left corner of that window. In this diagram, both sheets contain only one channel, as this is the most common configuration used throughout the thesis.

Finally, the input neuronal sheet is zero-padded to the physical size of at least $H_{\mathrm{out}} + 2r$, which is required to support all output neurons together with their connection radius r.

[3]Specifying two out of these properties implies the value of the third.

Because the neuronal sheets are defined in physical units, layer also calculates a distance tensor $\mathsf{D} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times k_{\text{h}} \times k_{\text{w}}}$, containing the physical distance of that artificial synapse. This can be used, for example, to disable connections beyond the selected radius r, creating a circular connection field.

While this method is useful for closely matching biologically motivated neuronal sheets, it is unnecessarily complex for standard deep learning architectures. For such cases, the kernel size is specified directly, and the neuronal sheets are either aligned directly on top of each other and zero-padded (`padding="same"`), or the input sheet is made just large enough to allow perfect non-padded connectivity (`padding="valid"`). The padding modes are visualized in Figure 4.2.



(a) (b)

Figure 4.2: Comparing valid and same padding in `LocalLinear` layers. **(a)** The "same" padding mode keeps the neuronal sheets exactly aligned. **(b)** The "valid" padding mode enlarges the input sheet just enough to support the selected size).

To further simplify the use of this layer, the framework provides lazy variants, which infer their input dimensions automatically during the first forward pass.

4.1.2 Weight Initialization

To initialize the kernel weights K, we use the *Kaiming uniform* method proposed by He et al. (2015b), sampling the weights from the uniform distribution $U\left(-\text{gain} \cdot \sqrt{\frac{3}{\text{fan\_in}}}, \text{gain} \cdot \sqrt{\frac{3}{\text{fan\_in}}}\right)$, where $\text{fan\_in} = C_{\text{in}} \cdot k_{\text{h}} \cdot k_{\text{w}}$ is the number of inputs contributing to a single output, and gain depends on the used nonlinearity (for ReLU, $\sqrt{2}$). The bias is initialized to zero.

4.1.3 Custom CUDA Implementation

While it is in principle possible to define this layer purely in PyTorch —for example using a masked fully connected layer—such naive approaches require enormous amounts of memory. Connecting two sheets of size 160×160 containing a single channel requires 160^4 floats (approximately 2.4 GiB of memory) to store all the weights. If weights are kept in a compact tensor of shape $(H_{\text{out}} W_{\text{out}} C_{\text{out}} C_{\text{in}} H_{\text{in}} W_{\text{in}})$, they only require $160^2 \cdot k_{\text{h}} k_{\text{w}}$, which for a kernel size of 17 results in only 28.2 MiB.

However, standard high-level operations like `torch.unfold()` and `torch.index_select()` do not fully solve the problem. While these operations create memory-efficient views, they result in non-contiguous tensors. When these tensors are passed to subsequent matrix multiplication routines, the system forces an expansion into contiguous memory to perform the computation. This leads to similar memory requirements as masked fully-connected approach.

To circumvent this limitation, we developed a custom CUDA kernel for the `LocalLinear` layer. This implementation maintains the memory-compact weight representation and utilizes the parallel processing capabilities of the GPU to perform the forward and backward steps while maintaining a minimal memory footprint.

In the forward pass, each CUDA thread is assigned a single output coordinate (b, i, j, o) to compute:

$$\text{LL}(\mathsf{I}; \mathsf{K})_{b,i,j,o} = \sum_{m=0}^{k_{\text{h}}-1} \sum_{n=0}^{k_{\text{w}}-1} \sum_{c=0}^{C_{\text{in}}-1} \mathsf{I}_{b,\phi(i)+m,\phi'(j)+n,c} \mathsf{K}_{i,j,o,c,m,n} \,,$$

where b is the position within batch. An additional version of this CUDA thread is provided which utilizes shared memory to store the relevant subset of weights, $\mathsf{K}_{i,j,o,*,*,*}$. This version is employed automatically when the batch size exceeds the empirically selected threshold of 16. The bias B is added separately, as it can be handled by standard library operations without a specialized kernel. For ϕ and ϕ', we use the previously mentioned lookup tables.

To backpropagate the incoming gradients $\mathsf{G} \in \mathbb{R}^{B \times H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}}$ back into the weights, a CUDA thread is launched for each weight $\mathsf{K}_{i,j,o,c,m,n}$, calculating the gradient as a sum over the batch of size B:

$$\frac{\partial \mathcal{L}}{\partial \mathsf{K}_{i,j,o,c,m,n}} = \sum_{b=0}^{B-1} \mathsf{I}_{b,\phi(i)+m,\phi'(j)+n,c} \mathsf{G}_{b,i,j,o} \,.$$

Lastly, the gradients of inputs are computed as:

$$\frac{\partial \mathcal{L}}{\partial \mathsf{I}_{b,c,y,x}} = \sum_{i=\alpha(y)}^{\zeta(y)} \sum_{j=\alpha'(x)}^{\zeta'(x)} \sum_{o=0}^{C_{\text{out}}-1} \mathsf{K}_{i,j,o,c,y-\phi(i),x-\phi'(j)} \mathsf{G}_{b,i,j,o} \,,$$

where $\alpha(y)$ returns the first i such that there exists a weight going from $\mathsf{I}_{\_,y,\_,\_}$ to $\mathsf{O}_{\_,i,\_,\_}$. Here, the _ denotes an arbitrary index. Similarly, $\zeta(y)$ is the last such i. α' and ζ' are defined analogously for x and j. Practically, the α and ζ functions are implemented using lookup tables.

To allow for backpropagation, the forward step stores a reference to the used weight K, input tensor I, as well as the lookup tables required for calculating ϕ, ϕ', α, α', ζ, and ζ'.

4.2 Hypernetwork Parameterization

Inspired by the hypothesis that biologically meaningful connection fields occupy a structured low-dimensional subset of the high-dimensional space of all possible local connection fields, and further motivated by the need to reduce the total number learnable parameters, we develop a hypernetwork parameterization of the `LocalLinear` layer. In this parameterization, each neuron is assigned a learnable parameter vector that serves as a compact descriptor from which pairwise connection weights can be generated.

Formally, for each neuronal sheet of shape (H, W, C), we introduce a learnable parameter tensor $\mathsf{N} \in \mathbb{R}^{H \times W \times C \times P}$, where P denotes the dimensionality of the neuronal descriptor. For a given connected pair of neurons, the corresponding connection weight is generated by a hypernetwork $\mathcal{H}$ from the tuple $(\Delta x, \Delta y, \boldsymbol{p}_1, \boldsymbol{p}_2)$, where $(\Delta x, \Delta y)$ is the relative spatial offset between the neurons, and $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{R}^P$ are their respective descriptors.

In the implementation, each hypernetwork-parameterized `LocalLinear` layer uses `torch.nn.utils.parametrize` to register a parameterization module `HyprWeightLL` for its kernel tensor. During each forward pass, the kernel tensor $\mathsf{K} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}} \times C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}}$ is generated using the neural network illustrated in Figure 4.3. This can be written as:

$$\mathsf{K}_{i,j,o,c,m,n} = \mathcal{H}\left(\Delta x, \Delta y, \boldsymbol{p}_{y,x,c}, \boldsymbol{p}_{i,j,o}\right),$$

where the output sheet's coordinates are denoted by i, j, input sheet coordinates by x, y, and the kernel coordinates corresponding to this connection m, n. The vectors $\boldsymbol{p}_{y,x,c}$ and $\boldsymbol{p}_{i,j,o}$ are their respective neuronal descriptors, while o and c denote the output and input channels, respectively. These indices are illustrated in Figure 4.4.

To generate the entire kernel tensor at once, a tensor of concatenated relative offsets and neuronal descriptors is constructed:

$$\mathsf{D} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}} \times C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}} \times (2+P_1+P_2)},$$

where P_1 and P_2 denote the dimensionalities of the input and output neuronal descriptors. This tensor is then reshaped to:

$$\mathsf{D}' \in \mathbb{R}^{(H_{\text{out}} \cdot W_{\text{out}} \cdot C_{\text{out}} \cdot C_{\text{in}} \cdot H_{\text{in}} \cdot W_{\text{in}}) \times (2+P_1+P_2)} = \mathbb{R}^{N \times (2+P_1+P_2)},$$

and passed through the hypernetwork $\mathcal{H}$, yielding an output of shape $\mathbb{R}^N$. This result is finally reshaped to obtain the kernel tensor:

$$\mathsf{K} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}} \times C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}}.$$

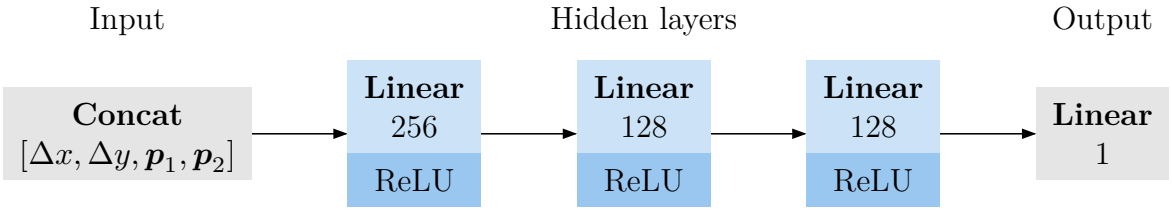


Figure 4.3: Architecture of the weight-generating hypernetwork. The model takes the relative spatial offset $(\Delta x, \Delta y)$ between neurons n_1 and n_2 and concatenates it with their respective parameter vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. This input is processed through a series of linear layers with ReLU activations to produce the final connection weight.

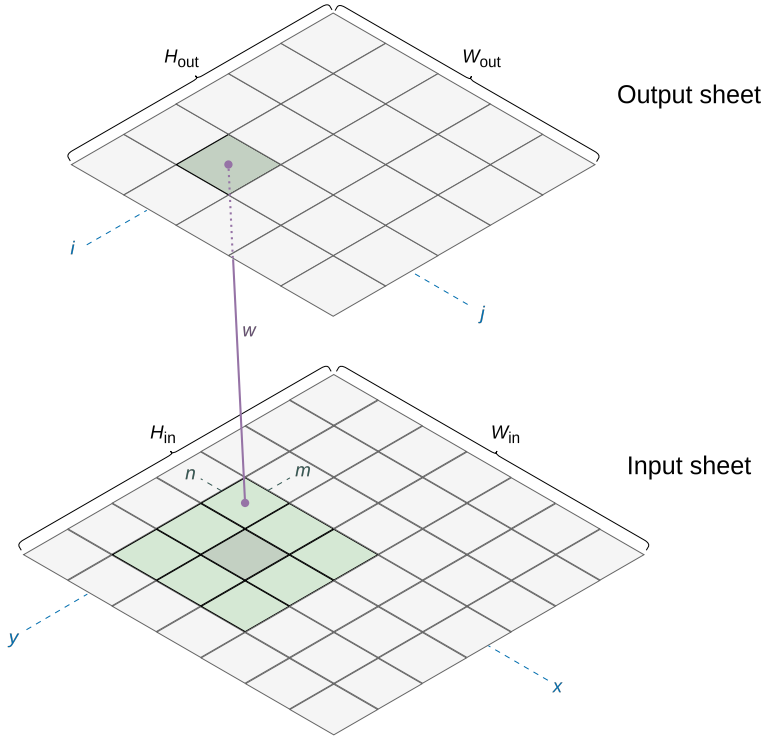


Figure 4.4: Indices used in the hypernetwork formulation. Here, i, j denote the coordinates in the output sheet, x, y denote the coordinates in the input sheet, and m, n denote the kernel coordinates associated with the connection. For clarity, only a single channel is illustrated.

The neuronal descriptors are sampled from $\mathcal{N}(0, 1)$, while the spatial offsets $\Delta x, \Delta y$ are rescaled to lie in the interval $(-1, 1)$. The implementation also supports configurations in which the two-dimensional spatial offset $(\Delta x, \Delta y)$ is replaced by a single distance scalar d, rescaled to lie in the interval $(0, 1)$.

4.2.1 Memory Optimization

In a naive implementation, the hypernetwork introduces a significant memory bottleneck because gradient computation requires retaining all input and intermediate tensors from the hypernetwork forward pass. For the hypernetwork from Figure 4.3, assuming $P_1 = P_2 = 3$, this amounts to $8 + 256 + 128 + 128 = 520$ additional floats per generated weight. For an output sheet of size 160×160 and a kernel of size 17×17, this corresponds to approximately 14.3 GiB of additional memory per layer. This memory cost grows quadratically with both output-sheet size and kernel size.

To reduce this cost, we introduce two complementary improvements. First, we implement a two-step backpropagation procedure. After the weights are generated, they are detached from the computational graph, so the hypernetwork activations can be discarded. During the backward pass, the hypernetwork forward computation is recomputed and gradients are then propagated to the hypernetwork parameters. This is conceptually similar to recomputation-based memory optimization strategies used in prior work, for example in Vančura et al. (2024).

Second, both the forward and backward hypernetwork computations are performed in chunks of size M, rather than generating the entire kernel at once, which further reduces peak memory usage. This is only effective because the generated weights are detached from the computational graph; otherwise, activations from all chunks would still be retained.

In the current implementation, M set to $4\,000\,000$ by default and reduced when additional memory savings are required. Smaller values of M lower peak memory usage but increase runtime due to the larger number of required hypernetwork evaluations. At the same time, the reduced memory footprint allows for larger training batch size, which can improve the overall training throughput. The choice of M is therefore subject to case-by-case optimization.

Together, these two improvements replace a large per-layer memory overhead with a depth-invariant peak memory requirement determined primarily by the chunk size M and the hypernetwork itself.

4.2.2 Learning-Rate Scaling of Neuronal Descriptors

We observe that, under the default optimization settings, the neuronal descriptors change only minimally from their randomly initialized values. Increasing their learning rate addresses this issue and leads to improved topographic organization as well as better image classification performance. In practice, this scaling factor is typically set between 100 and 1000.

4.2.3 Hypernetwork Initialization

Before training the full model, we pretrain the weight-generating hypernetwork as an initialization step. The goal is to make the generated connection weights approximately zero-mean with a distance-dependent variance. We define a target variance for each connection based on its spatial distance and normalize these variances by a global scale factor so that their average over the layer's connection grid is one. During pretraining, we sample random batches of neuronal descriptors and spatial offsets, generate the corresponding weights, and divide them by the square root of their target variance. If the initialization matches the target distribution, the normalized weights should approximately follow $\mathcal{N}(0,1)$. We therefore estimate their empirical mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$, and minimize $D_{\mathrm{KL}}(\mathcal{N}(0,1) \parallel \mathcal{N}(\hat{\mu}, \hat{\sigma}^2))$.

4.3 SIREN-Based Parameterization

Our hypernetwork can already be interpreted as an implicit neural representation (INR) of the connection fields, parameterized by the neuronal descriptors. Inspired by (Sitzmann et al., 2020)—which shows that sinusoidal activation functions can improve convergence speed and representation fidelity—we incorporate SIREN-style parameterization into our framework.

A SIREN network can be understood as a standard multilayer perceptron in which ReLU activations are replaced by sinusoidal activations. It also uses a specialized weight initialization. In this thesis, we follow the authors' reference

implementation, in which the first-layer weights are sampled from $U\left(-\frac{1}{n}, \frac{1}{n}\right)$, while in the deeper-layers weights are sampled from $U\left(-\frac{1}{30}\sqrt{\frac{6}{n}}, \frac{1}{30}\sqrt{\frac{6}{n}}\right)$.

We introduce coordinate scaling as an additional hyperparameter to SIREN, allowing control over the range of coordinates presented at the network's input. We observe that this parameter has a strong effect on convergence speed when SIREN is used to learn image representation directly.

First, we apply SIREN-style parameterization directly in the weight-generating hypernetwork. In this setting, the coordinate scaling parameter is set to 20, as empirical experiments showed that this value allows the network to represent Gabor-like patterns reliably. The resulting architecture is illustrated in Figure 4.5.

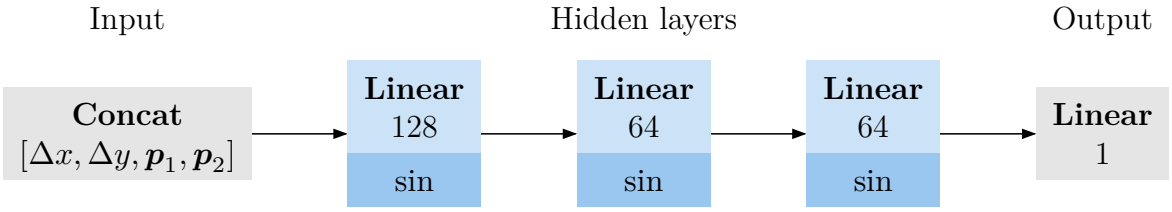


Figure 4.5: Architecture of the SIREN-based weight-generating hypernetwork. This network has the same overall structure as Figure 4.3, but replaces ReLU activations with sinusoidal activations. The hidden-layer widths are halved, as in our experiments, this smaller variant already performs better than the original ReLU-based hypernetwork.

Second, we utilize a SIREN network to generate the sheet of neuronal descriptors. Formally, this network is represented as a function $\mathcal{S} : \mathbb{R}^2 \to \mathbb{R}^{CP}$, where P is the dimensionality of each neuronal descriptor and C is the number of channels. The network maps a two-dimensional coordinate (x, y) on the neuronal sheet to the corresponding set of neuronal descriptors at that location, one for each channel. The architecture of this model is illustrated in Figure 4.6.

The motivation for this parameterization is to encourage spatial regularity in the learned descriptor sheets, and consequently in the resulting connection fields, without imposing the rigid weight sharing used in traditional convolutions.

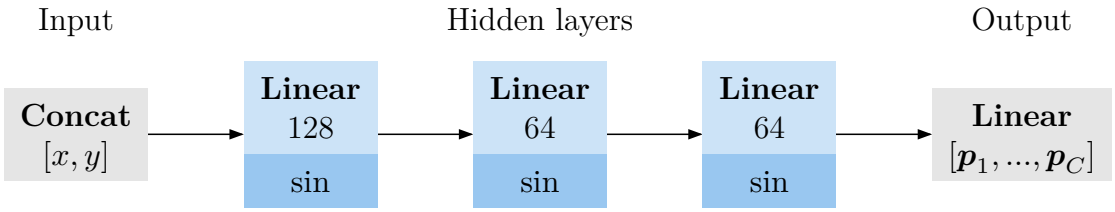


Figure 4.6: Architecture of the SIREN-based neuronal-descriptor network. This network takes a two-dimensional coordinate (x, y) on the neuronal sheet as input and outputs the corresponding set of neuronal descriptors, one for each channel. Because of the sinusoidal activations, this provides a smooth representation of the neuronal descriptor sheet.

For this network, the coordinate scaling is set to 50, as empirical experiments showed that this value allows the network to represent smooth structures resembling biological orientation preference maps, thereby providing a suitable proxy for the expected spatial scale of global organization in the model[4].

The learning-rate scaling hyperparameter needs to be set differently than for non-SIREN neuronal descriptor sheets, as the sinusoidal parameterization exhibits different optimization dynamics. In our experiments a value of approximately 5 to 10 leads to stable training and smooth map formation.

4.4 Network Architecture

4.4.1 LocalLinear Models

The main experiments use variants of the `LocalLinear`-based architecture illustrated in Figure 4.7. Across the main comparisons, the overall architecture is kept fixed, and only the parameterization mechanism used to generate the `LocalLinear` weights is varied.

A defining feature of this architecture is that it uses a single channel throughout, which allows direct inspection of neuronal connection fields and population-level topographic organization, such as orientation preference maps. The spatial sizes of the layers are loosely inspired by the relative scaling of cortical areas in macaque visual cortex, as described by DiCarlo et al. (2012).

For the hypernetwork-parameterized variants, each `LocalLinear` layer is additionally associated with a sheet of neuronal descriptors. These descriptors are not attached to the input image itself, but only to the intermediate neuronal sheets. The neuronal sheets associated with the architecture in Figure 4.7 are illustrated in Figure 4.8.

In total, we implement four main variants of our model:

- `LL`: the baseline `LocalLinear` model, in which weights are learned directly without hypernetwork parameterization.
- `LL-Hypr`: a hypernetwork parameterized `LocalLinear` model, in which each neuronal sheet is associated with a directly learned sheet of neuronal descriptors.
- `LL-HyprSiren`: the same hypernetwork-parameterized model, but with the weight-generating hypernetwork utilizing SIREN's sinusoidal activations.
- `LL-FullSiren`: the full SIREN-parameterized model, in which the hypernetwork uses sinusoidal activations and the neuronal descriptor sheets are themselves generated by a SIREN network.

[4]That is, the sheet coordinates are rescaled so that $x, y \in [-50, 50]$ over the spatial domain represented to the network.

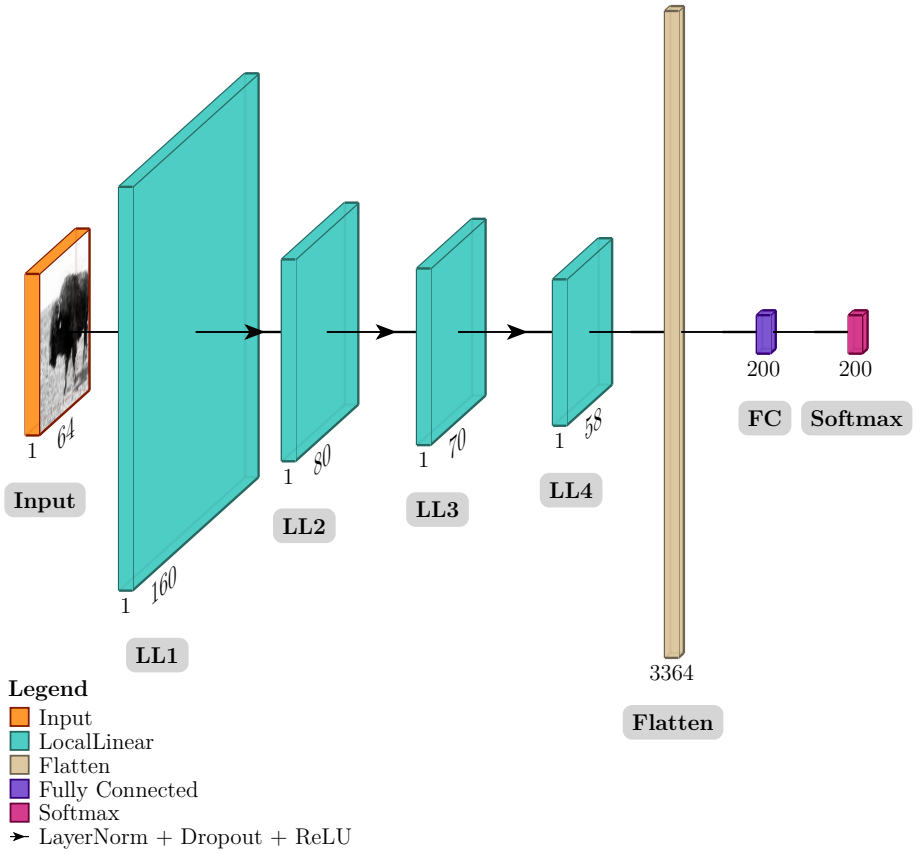


Figure 4.7: The model used in the main experiments, utilizing the `LocalLinear` layers.

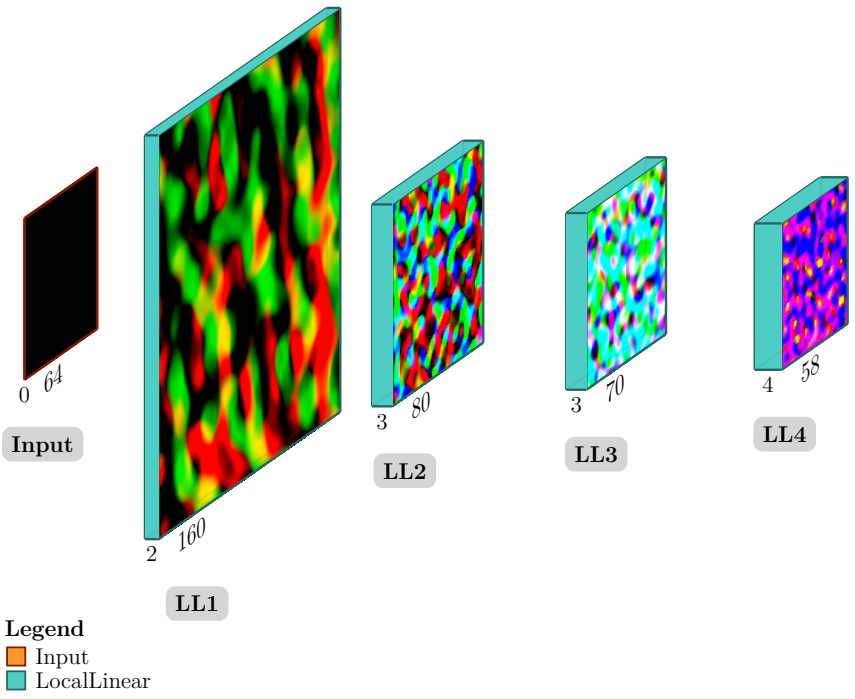


Figure 4.8: Neuronal descriptor sheets assigned to the input and `LocalLinear` layers. Individual descriptors are mapped to the RGB space. If the descriptor has more than three values, only first the three are shown. These particular neuronal descriptor sheets were parameterized by the SIREN network.

4.4.2 Convolutional Baselines

To evaluate the proposed architectures against standard convolutional models, we compare them with two types of baselines: a modified ResNet-18 and a set of simpler convolutional networks designed to more closely match the proposed model in architecture and computational cost.

As a strong reference model, we use a modified ResNet-18 (He et al., 2015a). Because Tiny ImageNet images in our experiments have resolution $64 \times 64 \times 1$, which is smaller than inputs used by the original formulation, several adjustments are made. The initial 7×7 convolution followed by 3×3 max pooling is replaced by a single 5×5 convolution with 64 output channels and no max pooling. We also use "same" padding throughout and replace the original classification head with the same classification head used in the other experiments.

In addition to this strong baseline, we construct simpler convolutional models, named `CNN1` and `CNN2`, intended to provide fairer comparisons to the proposed architecture. These models are chosen to more closely match the `LocalLinear` network in terms of depth, parameter count, and floating-point operation count (FLOPs). The first of these models is illustrated in Figure 4.9. A summary of all compared architectures is provided in Table 4.1.

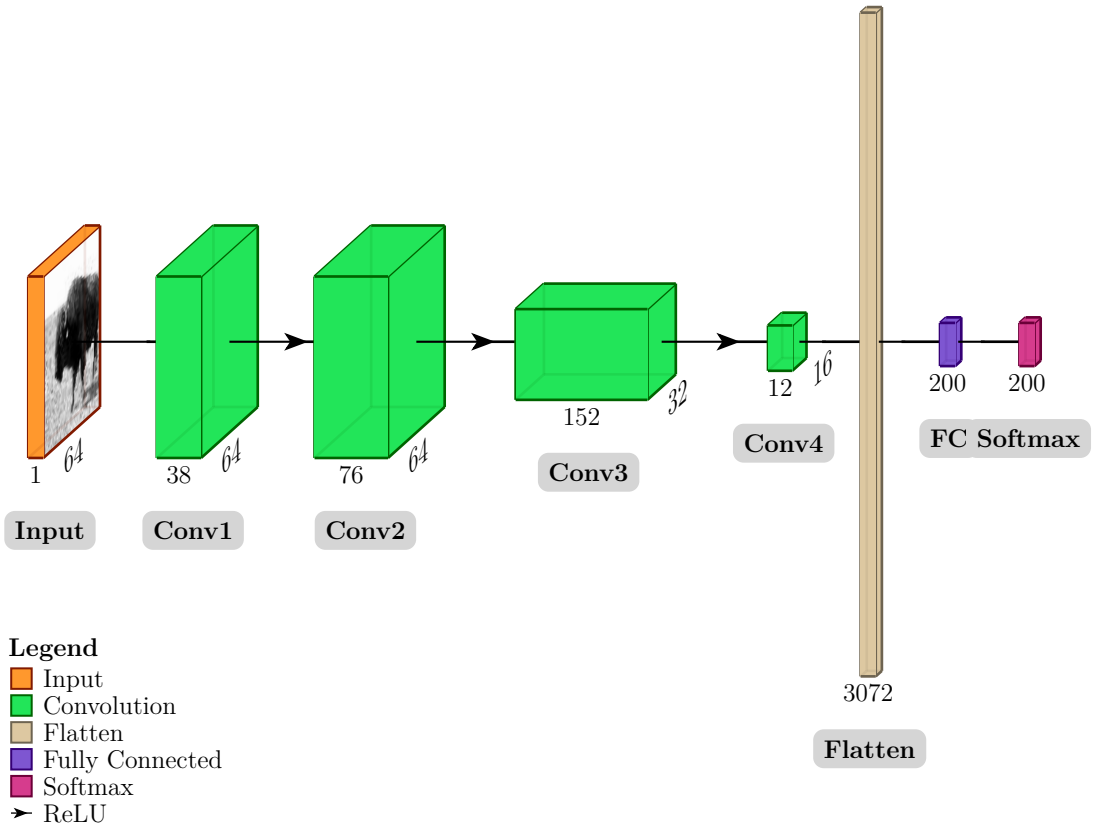


Figure 4.9: The convolutional baseline `CNN1`.

	CNN1			*CNN2*			*LL (ours)*		
	W	***C***	***K***	***W***	***C***	***K***	***W***	***C***	***K***
Input	64	1		64	1		64	1	
Layer1	64 (395)	38	5 (5)	64 (256)	16	5 (5)	160	1	15
Layer2	64 (558)	76	3 (18)	64 (256)	16	3 (12)	80	1	15
Layer3	32 (395)	152	3 (26)	32 (181)	32	3 (12)	70	1	15
Layer4	16 (55)	12	3 (37)	16 (55)	12	3 (17)	58	1	15

Table 4.1: Comparison of the proposed architecture with the two convolutional baselines. The table reports the width (W), number of channels (C), and the kernel size (K) for the baseline models `CNN1` and `CNN2` and our class of models. Values in parentheses indicate the equivalent single-channel `LocalLinear` width or kernel size required to match, respectively, the number of neurons or the number of input connections (rounded).

4.5 Parameter Counts and FLOPs

The computational comparisons in this thesis are based on trainable parameter counts and floating-point operation counts (FLOPs).

We define the parameter count as the number of trainable scalar values used to represent the weights—or their parameterizations—for all convolutional and `LocalLinear` layers. Bias parameters are excluded from this count.

We define FLOPs as the total number of scalar multiplications performed by all convolutional and `LocalLinear` layers during a forward pass. Equivalently, this corresponds to the number of artificial synapses (input-weight products) formed by these layers. Bias additions are excluded.

The parameter counts and FLOPs for all considered models are shown in Table 4.2.

Model	Parameters	FLOPs
`ResNet-18`	11.164 M	2221.15 M
`LL`	9.008 M	9.0 M
`LL-FullSiren`	0.364 M	9.0 M
`LL-Hypr`	0.152 M	9.0 M
`LL-HyprSiren`	0.152 M	9.0 M
`CNN1`	0.148 M	221.02 M
`CNN1`	0.011 M	16.7 M

Table 4.2: The parameter counts and FLOPs for the `LocalLinear` models as well as convolutional baselines.

4.6 Visualizations

Three main visualizations are used throughout the thesis: connection fields, neuronal descriptor sheets, and orientation preference maps (OPMs).

Connection fields are visualized by selecting the neurons nearest the centers of a 20×20 grid placed over the neuronal sheet. The resulting image makes it possible to inspect both connection fields of individual neurons as well as their global spatial organization. Positive values are mapped to the green channel and negative to the red channel.

Neuronal descriptor sheets are visualized directly by mapping the first three components of each neuronal descriptor to red, green, and blue channels, respectively. Any additional descriptor dimensions are ignored in the visualization.

To obtain orientation preference maps, we present the model with sinusoidal gratings spanning 8 different orientations and 8 different phases. Example gratings are illustrated in Figure 4.10. For each neuron, responses are averaged across phases for each orientation. The preferred orientation is then estimated by computing a vector average over the orientation responses using doubled angles, which makes orientations separated by 180° equivalent and causes orthogonal orientations to cancel out. The resulting image is mapped back to the initial 0° to 180° interval, color-coded, and assigned to the neuron's position on the sheet to form the final map.

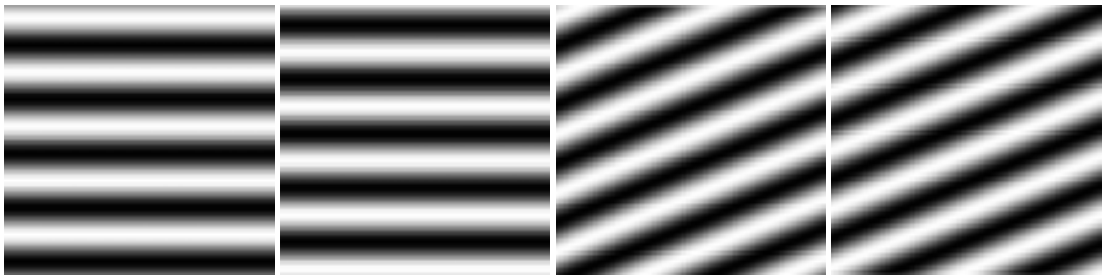
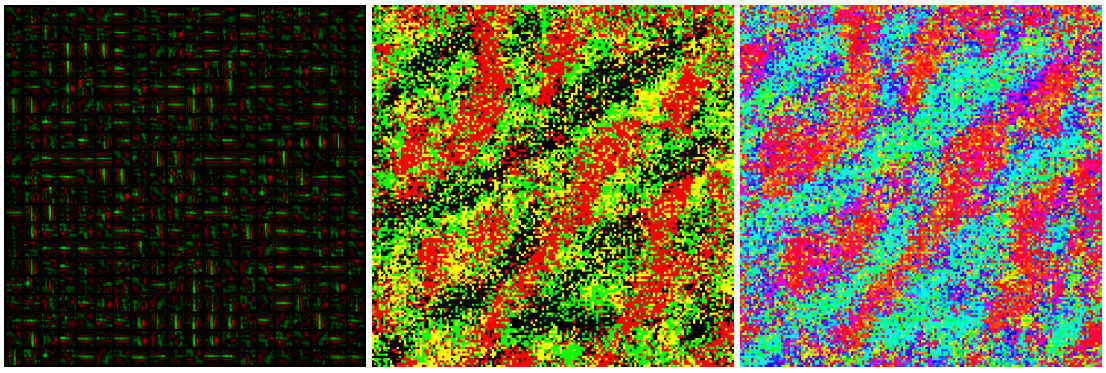


Figure 4.10: Example sinusoidal gratings used to estimate orientation preference maps. Two figure shows two orientations, each at two different phase offsets.

An example of all visualization types is shown in Figure 4.11.



(a) (b) (c)

Figure 4.11: Example visualizations. **(a)** connection fields, **(b)** neuronal descriptor sheet, **(c)** orientation preference map (each color represents a different stimulus orientation preference).

4.7 Dataset and Task

We train the proposed models for image classification on the Tiny ImageNet dataset (Wu et al., 2017), which contains 200 image classes with 500 training examples per class. All images have resolution $64 \times 64 \times 3$. To simplify the analysis of emerging connection fields, all images are converted to grayscale before training.

Because our `LocalLinear` does not impose the rigid translational weight sharing characteristic of standard convolutional layers, we apply small random-roll augmentation during training. With probability 25%, each image is shifted by a random offset of up to 25% of the image size along either the horizontal or vertical axis.

4.8 Training Setup and Hyperparameters

We train our models for 120 epochs using the Adam optimizer (Kingma & Ba, 2017) with a constant learning rate of 10^{-4}. When neuronal descriptor sheets are present, their learning rate is scaled by a factor of either 1000 for directly learned neuronal descriptor, or 10 for SIREN-based neuronal descriptors. The layers of the network are kept as illustrated in Figure 4.7 and Table 4.1, with the exception of `LL-Hypr` and `LL-FullSiren`, in which we find it performs better when we remove the ReLU activation function after the first `LocalLinear` layer. All hypernetworks used have widths [128, 64, 64]. The full set of training hyperparameters are summarized in Table 4.3.

Parameter	**Value**
Epochs	120
Learning rate	10^{-4}
Batch size	96
Dropout	$0.2^{\dagger}$
Weight decay	10^{-4}
Layer widths	[160, 80, 70, 58] (LL), [64, 64, 32, 16] (conv)
Kernel size	[15, 15, 15, 15] (LL), [5, 3, 3, 3] (conv)
Hypernetwork width	[128, 64, 64]
SIREN network width	[256, 128, 128]
Neuronal descriptor dimensions	[2, 3, 3, 4]
Neuronal descriptor LR scaling	$1000^{\dagger}$ (direct), 5 (SIREN)
Hypernetwork pretraining epochs	2000
Hypernetwork pretraining batch size	512

Table 4.3: Hyperparameters used for training. Values marked with † were selected based on preliminary comparisons reported in Appendix Section A.

5 Results

5.1 Connection Fields and Topography

We compare the four `LocalLinear` variants by examining how each parameterization affects local connection fields, orientation preference maps (OPMs), and, where applicable, neuronal descriptor sheets.

We begin with the `LL` model, which uses `LocalLinear` connections without any parameterization. As shown in Figure 5.1, its connection fields exhibit a characteristic "salt-and-pepper" pattern with little coherent structure. Here, "salt-and-pepper" refers to highly irregular structure in which neighboring connections appear scattered rather than smoothly organized. The corresponding OPMs in Figure 5.2 are similarly noisy and display only minimal global organization.

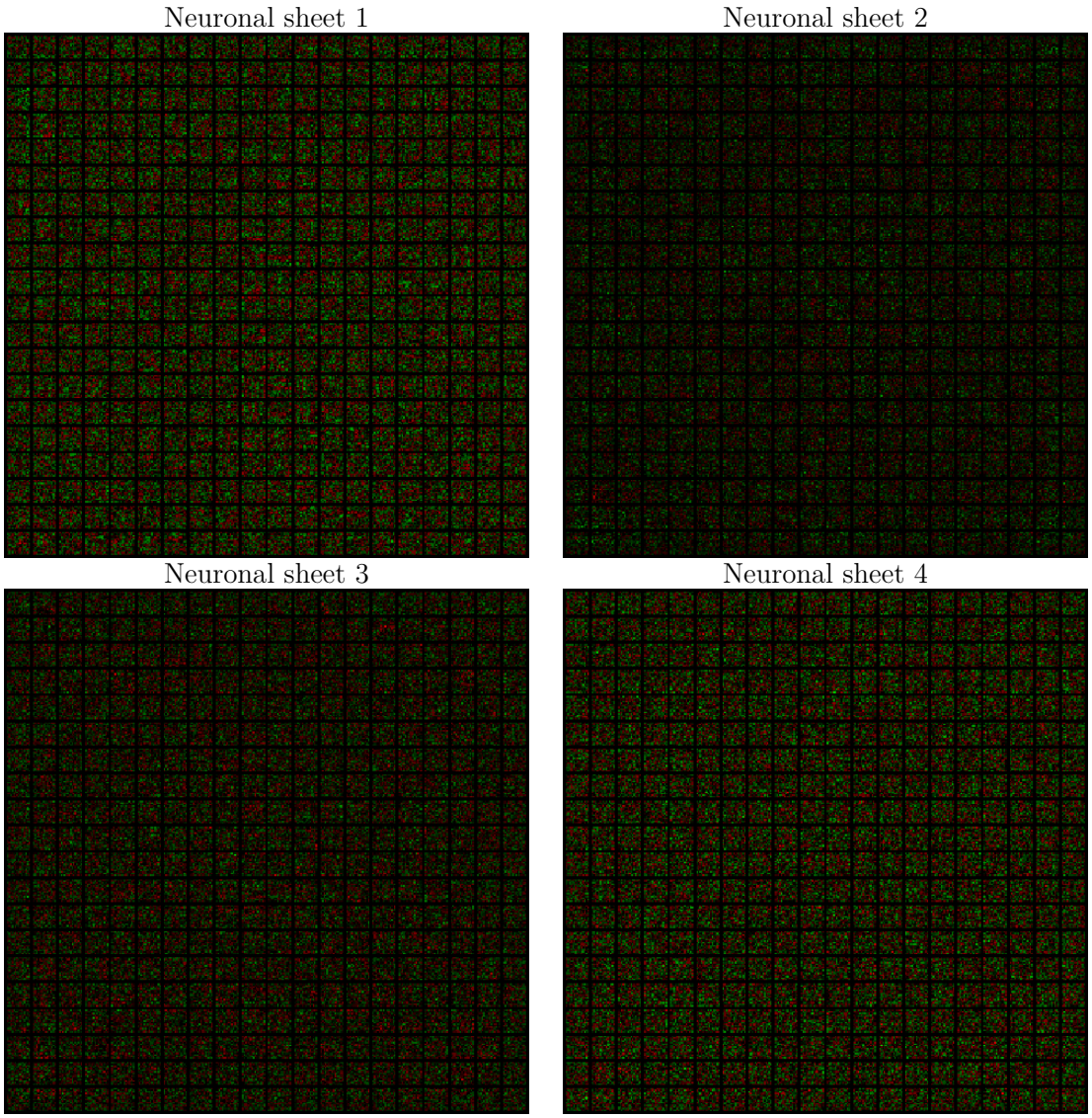


Figure 5.1: Connection fields of the `LL` model. The fields exhibit a characteristic "salt-and-pepper" pattern.

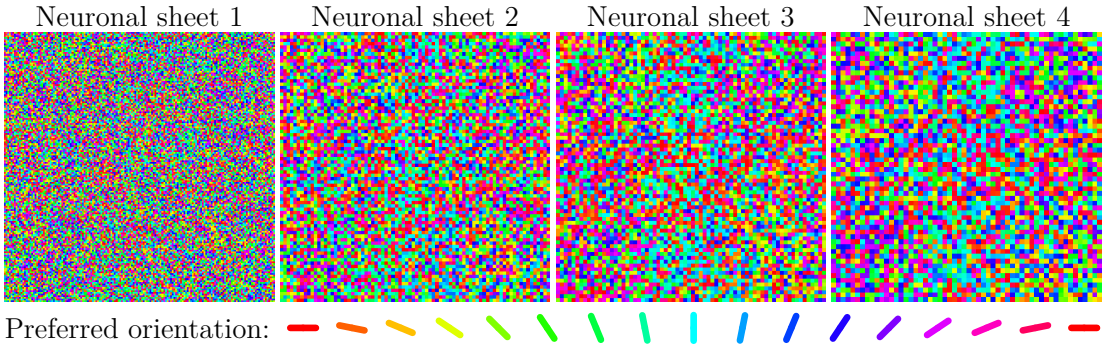


Figure 5.2: Orientation preference map of the `LL` model. The map is highly noisy and shows little population-level organization.

We next consider the `LL-Hypr` model, in which the `LocalLinear` weights are generated by a hypernetwork conditioned on neuronal descriptors and relative spatial offsets. As shown in Figure 5.5, the connection fields in the first layer become substantially more structured than in the plain `LL` model, with clearly oriented bars, but remain largely disorganized in the deeper layers. The corresponding neuronal descriptor sheets in Figure 5.3 appears very noisy with only weak signs of emerging organization. The resulting OPMs in Figure 5.4 are less noisy than those of LL, but still lack clear global structure. Compared with `LL`, the main effect of hypernetwork parameterization is the emergence of coherent local connection structure in the first layer.

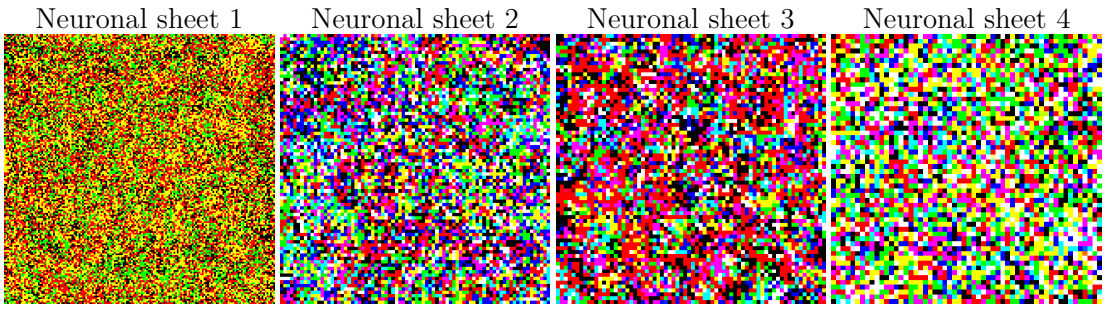


Figure 5.3: Neuronal descriptor sheet of the `LL-Hypr` model. Weak large-scale organization begins to emerge, but the sheets remain highly noisy.

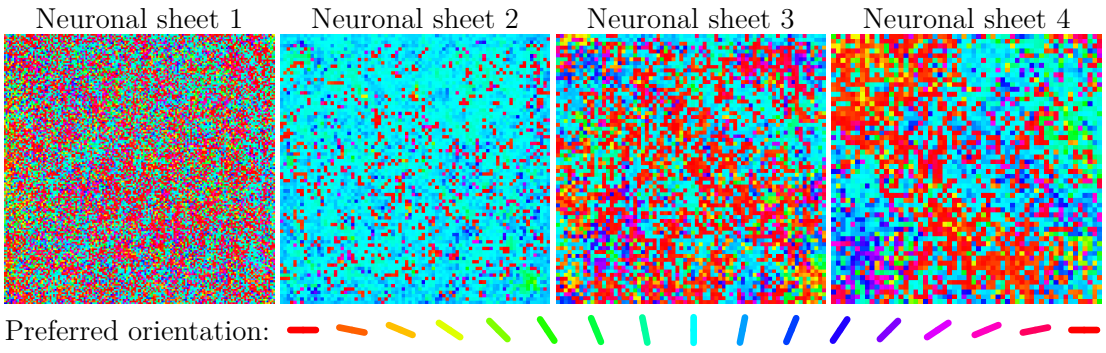


Figure 5.4: Orientation preference map of the `LL-Hypr` model. Some population-level organization emerges, but the maps remain highly noisy.

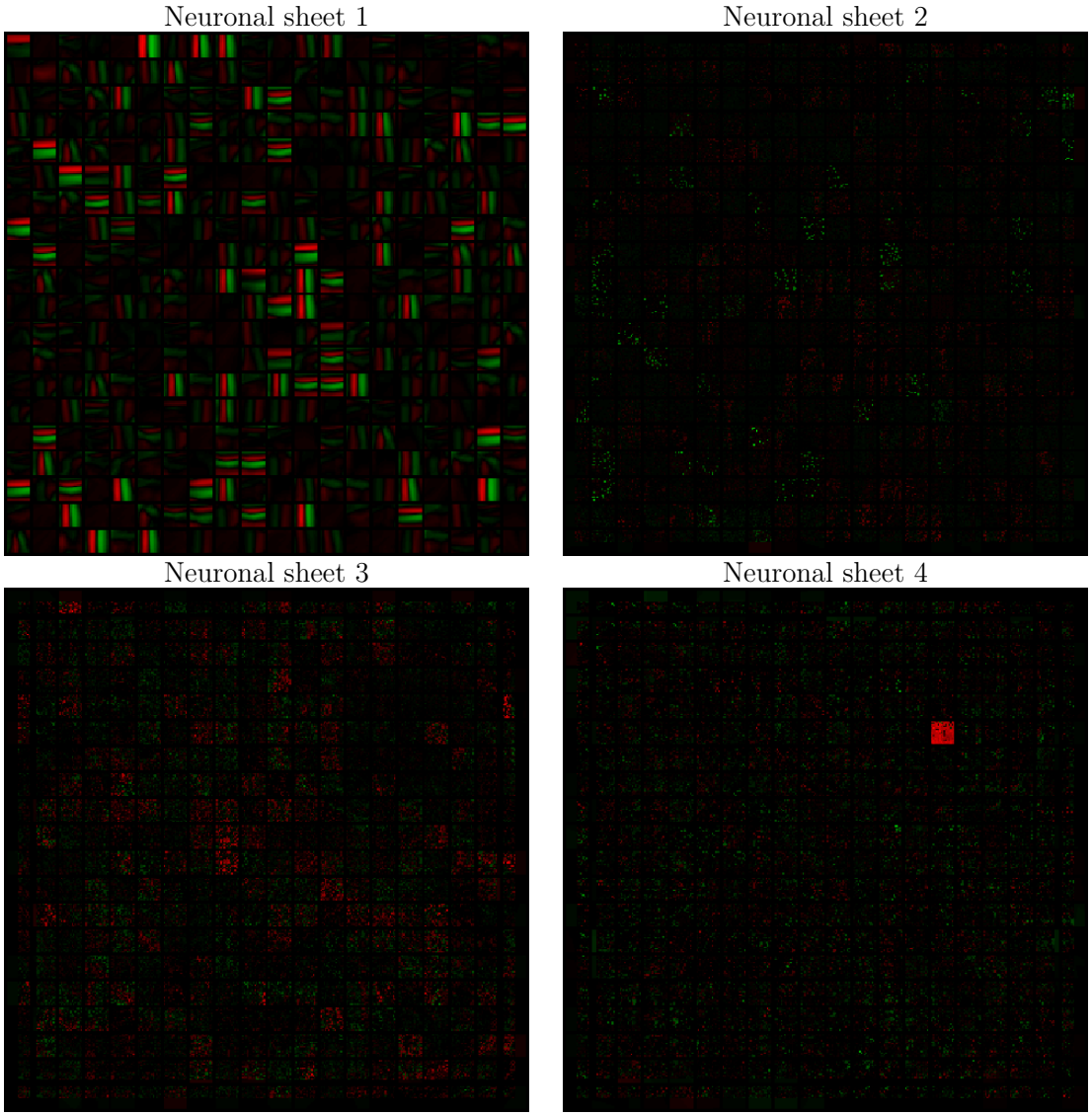


Figure 5.5: Connection fields of the `LL-Hypr` model. The first layer develops clearly oriented patterns, while the deeper layers remain larger salt-and-pepper.

Next, we consider the `LL-HyprSiren` model, in which the weight-generating hypernetwork uses SIREN-style sinusoidal activations. As shown in Figure 5.6, the connection fields become structured in all but last layer, rather than only in the first layer. The neuronal descriptor sheets in Figure 5.7 remain directly learned, but the improved hypernetwork parameterization leads to clearly organized regions in the first descriptor sheet. The OPMs in Figure 5.8 follow this organization closely and exhibit much clearer large-scale structure than in `LL-Hypr`. Compared with `LL-Hypr`, the main effect of the SIREN-based hypernetwork is the emergence of coherent topographic organization in the first layer together with structured connection fields across most layers.

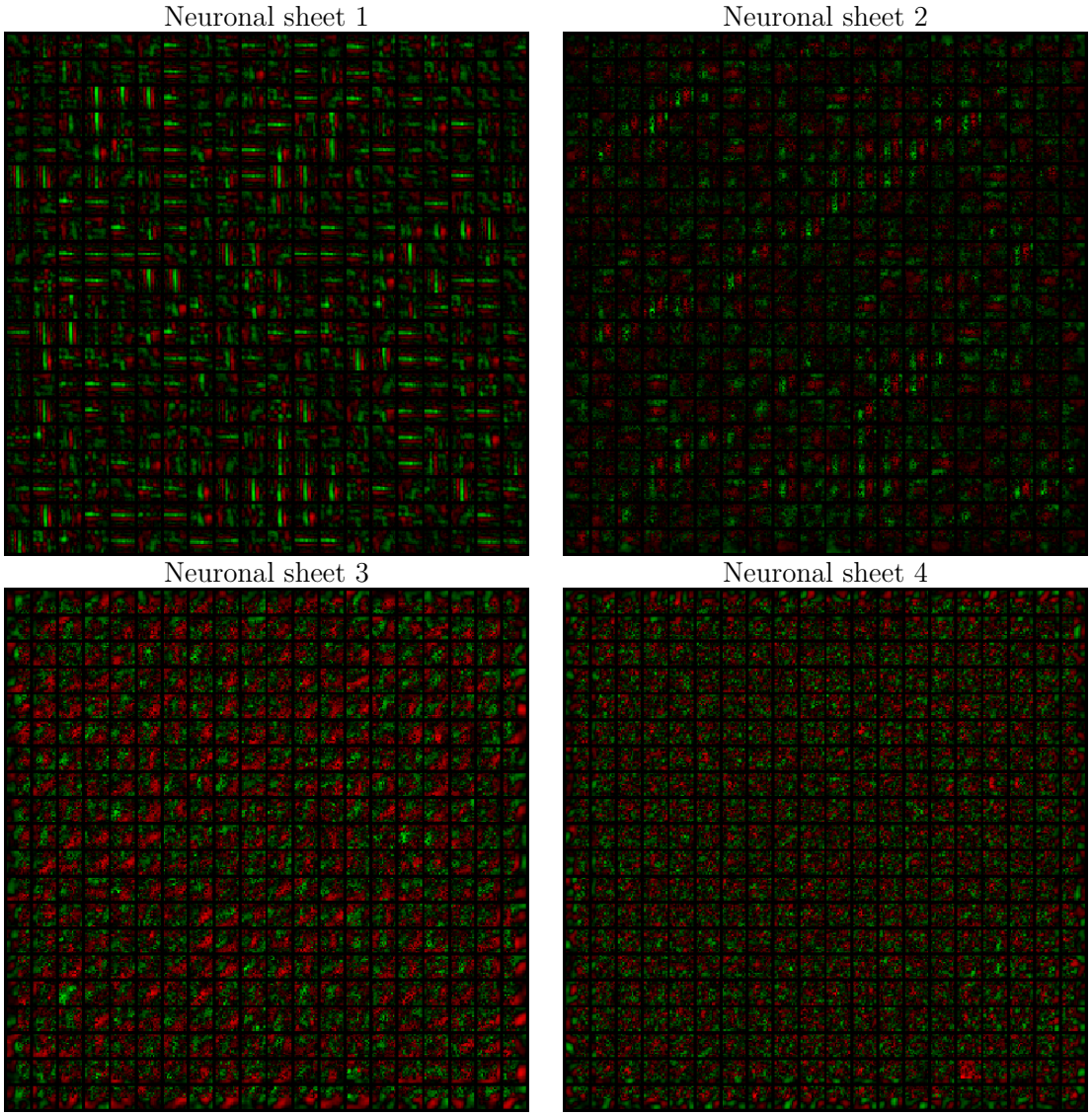


Figure 5.6: Connection fields of the `LL-HyprSiren` model. All layers, except for last, show structured and oriented connection fields.

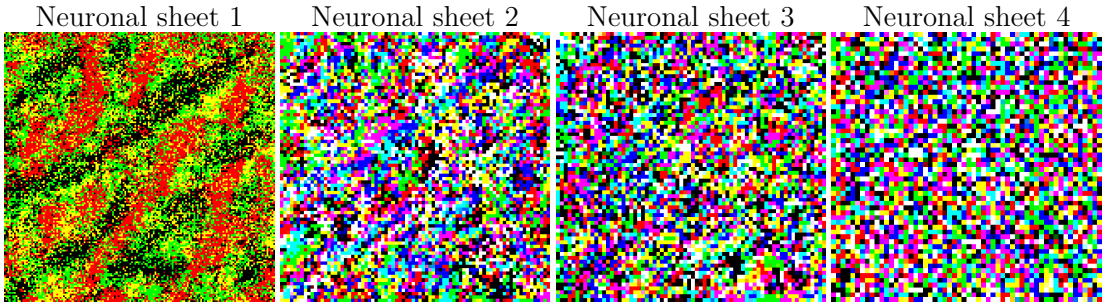


Figure 5.7: Neuronal descriptor sheet of the `LL-HyprSiren` model. The first layer shows clear separation into clusters lines.

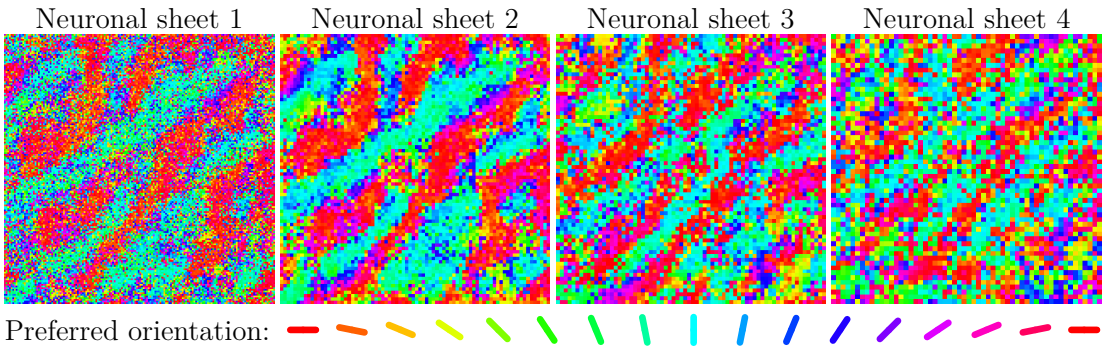


Figure 5.8: Orientation preference map of the `LL-HyprSiren` model. The map shows an organization into parallel-ish regions of the same orientation.

Finally, we consider the `LL-FullSiren` model, in which both the hypernetwork and the neuronal descriptor sheets are parameterized using SIREN-based implicit representations. As shown in Figure 5.11, the connection fields in the deeper layers are even more clearly structured than in `LL-HyprSiren`, with strong local regularity across all but the last layer. The neuronal descriptor sheets in Figure 5.9 form smooth spatial maps with global regularity. The corresponding OPMs in Figure 5.10 display clear smooth topographic structure across all layers, although some imperfections remain. Compared with `LL-HyprSiren`, the main effect of SIREN-based descriptor sheet parameterization is the emergence of coherent global organization in the deeper layers. The global organization of the first neuronal descriptor sheet remains similar to the `LL-HyprSiren`, indicating that the SIREN does not force the emergence of this structure.

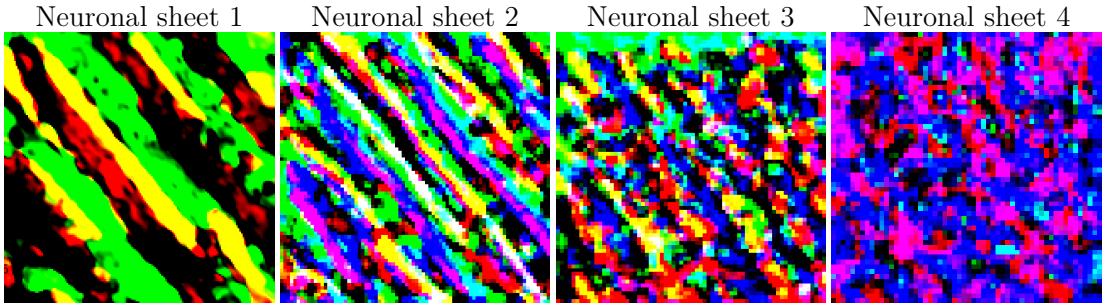


Figure 5.9: Neuronal descriptor sheet of the `LL-FullSiren` model. The sheets form smooth spatial maps, indicating globally organized descriptor structure.

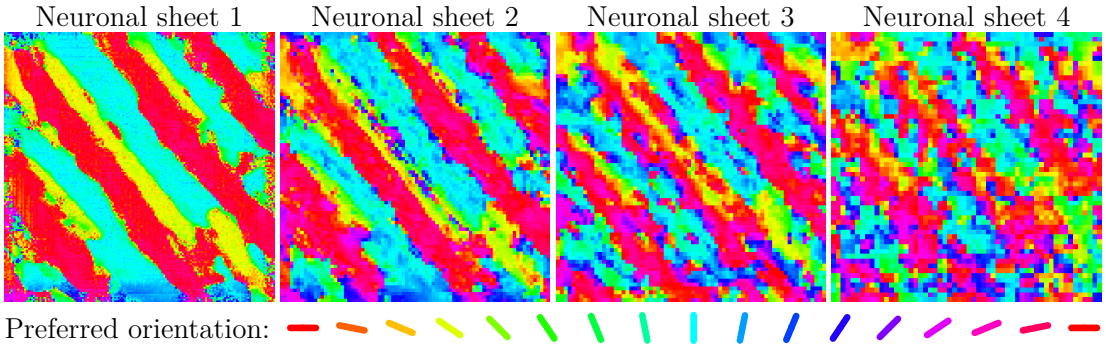


Figure 5.10: Orientation preference map of the `LL-FullSiren` model. The maps display clear smooth topographic organization across all layers, although some irregularities remain.

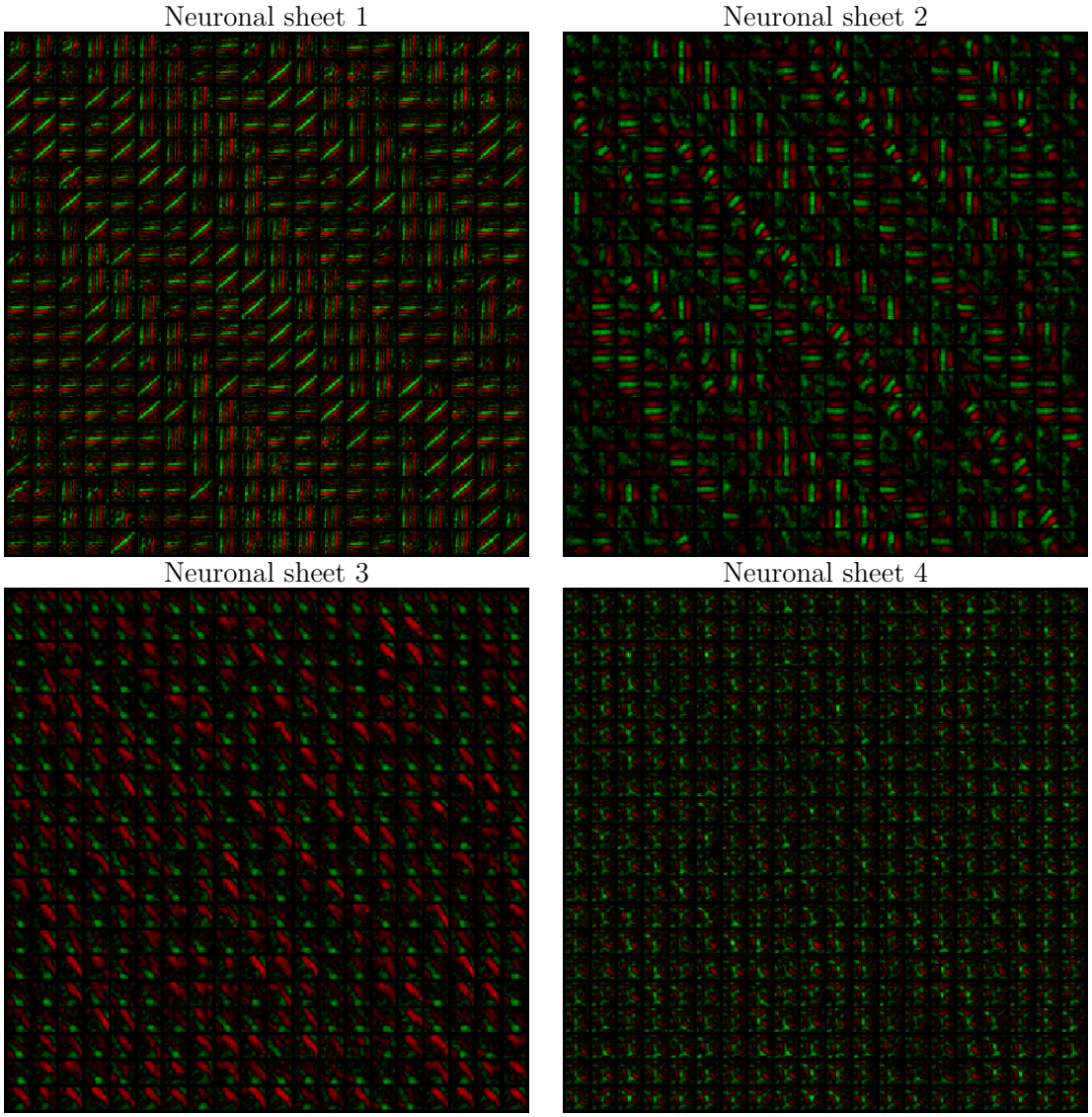


Figure 5.11: Connection fields of the `LL-AllSiren` model. The fields are highly structured and show local regularity across all but last layer.

5.2 Image Classification Performance

We evaluate image classification performance by comparing the proposed `LocalLinear` models with the convolutional baselines and the modified ResNet-18. Unless stated otherwise, each result is averaged over five runs.

As shown in Figure 5.12, our best-performing model, `LL-HyprSiren`, achieves higher performance than the baseline model `CNN2`. However, this improvement comes at the cost of substantially slower training. Whereas the convolutional baselines reach their peak performance on this task within approximately six epochs, `LL-HyprSiren` continues improving throughout the full 120 training epochs.

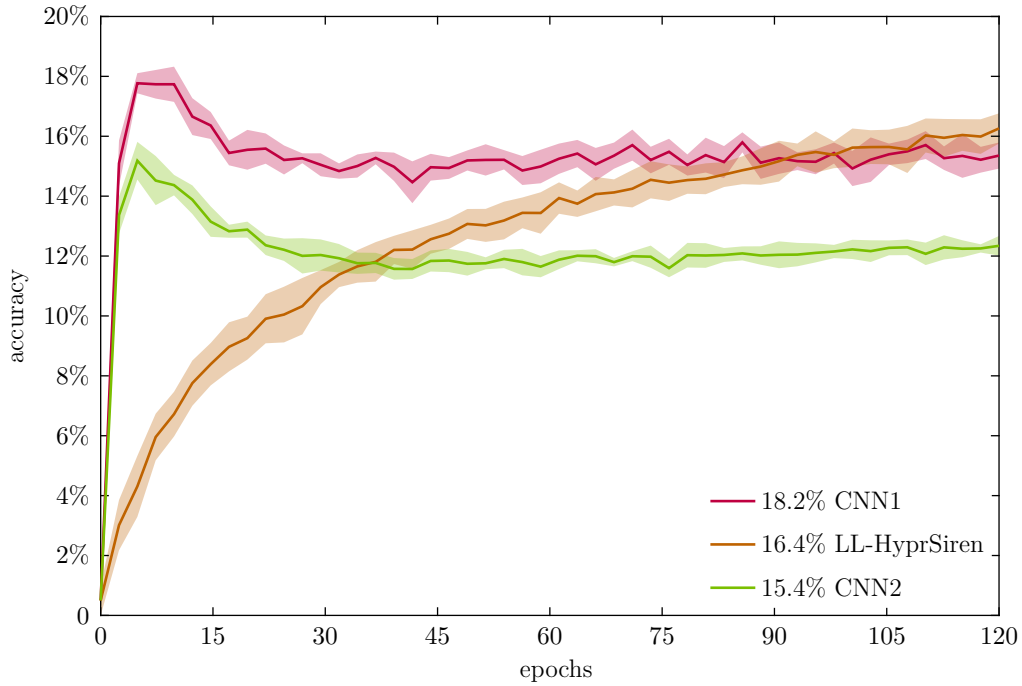


Figure 5.12: Training performance of the `LL-HyprSiren` model compared with the convolutional baselines. The graph shows mean accuracy over five runs at each training step, with the shaded region indicating standard deviation. The `LL-HyprSiren` model ultimately outperforms `CNN2`, but requires substantially longer training.

All trained models based on the `LocalLinear` layer are compared together in Figure 5.13. A particularly notable result is the improvement from `LL-Hypr` to `LL-HyprSiren`, which suggests that sinusoidal activations in the weight-generating hypernetwork are more effective than ReLU activations in this setting.

Also shown is the single run labeled `LL-FullSiren-Fluke`. This result was obtained prior to the final comparison experiments and uses a smaller SIREN network for generating neuronal descriptors, with hidden-layer widths [128, 64, 64] instead of the [256, 128, 128] used in the main LL-FullSiren experiments. This run suggests that `LL-FullSiren` is capable of reaching performance comparable to `LL-HyprSiren`. However, because this observation comes from a single run, it should be interpreted cautiously. It may indicate sensitivity to initialization or to the exact configuration of the descriptor-generating SIREN network.

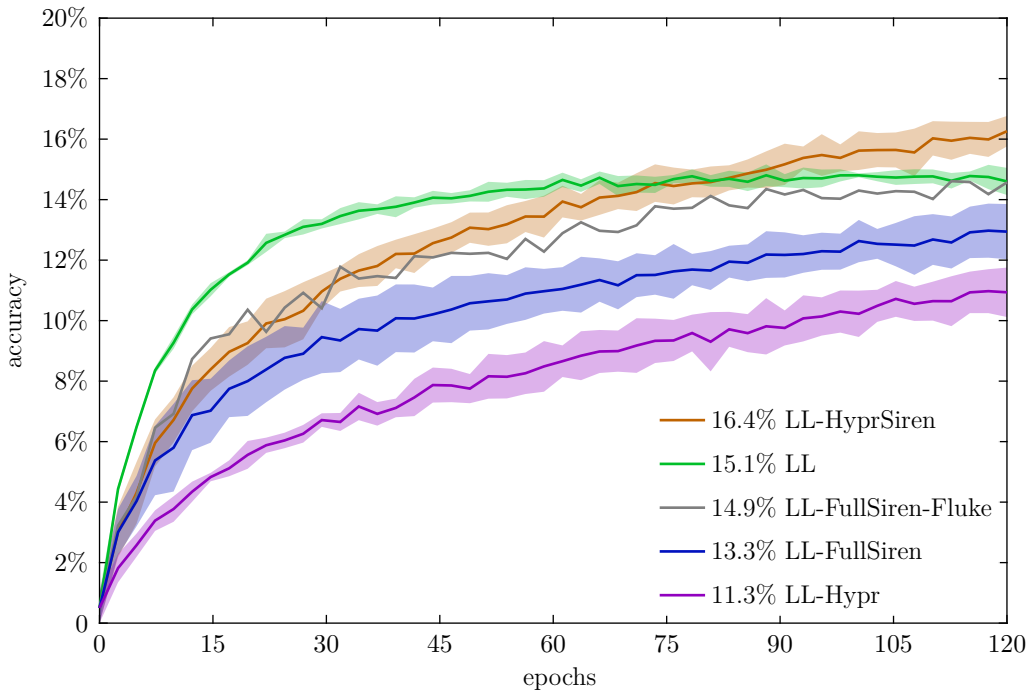


Figure 5.13: Training performance of all proposed `LocalLinear` models with different parameterizations. The graph shows mean accuracy over five runs at each training step, with the shaded region indicating standard deviation.

The average best accuracy of each model, together with the number of trainable parameters and FLOPs, is summarized in Table 5.1.

Model	Accuracy	Params	FLOPs
`ResNet-18`	47.1% ± 0.4%	11.164 M	2221.15 M
`CNN1`	18.2% ± 0.4%	0.148 M	221.02 M
`LL-HyprSiren`	16.4% ± 0.4%	0.152 M	9.0 M
`CNN2`	15.4% ± 0.4%	0.011 M	16.7 M
`LL`	15.1% ± 0.2%	9.008 M	9.0 M
`LL-FullSiren-Fluke`	14.9%	0.204 M	9.0 M
`LL-FullSiren`	13.3% ± 0.7%	0.364 M	9.0 M
`LL-Hypr`	11.3% ± 0.8%	0.152 M	9.0 M

Table 5.1: Summary of image classification performance, trainable parameter counts, and FLOPs for all compared models. The precise definitions of parameter count and FLOP count are given in the methods section (page 46).

5.3 Computational Scaling

When the weights of a `LocalLinear` layer are generated by a hypernetwork, the number of trainable parameters is independent of the kernel size. By contrast, for a standard convolutional layer, the number of trainable parameters scales with the kernel size, i.e. proportionally to $k_h k_w$.

The scaling of the neuronal descriptor sheets depends on how they are parameterized. If a descriptor sheet is learned directly, it requires $W \cdot H \cdot P$ trainable parameters, where P is the descriptor dimensionality. If the descriptor sheet is instead parameterized by a SIREN network, the number of trainable parameters becomes independent of the sheet size and is determined only by the size of the SIREN network itself.

In total, the `LL-FullSiren` model requires only $L(p_{\mathcal{H}} + p_{\mathcal{D}})$ trainable parameters, where L denotes the depth of the network, and $p_{\mathcal{H}}$ and $p_{\mathcal{D}}$ denote the number of parameters in a single weight-generating hypernetwork and a single SIREN neuronal descriptor network, respectively.

5.4 Memory Analysis

This section quantifies the memory savings introduced by the custom CUDA kernel and the two-step hypernetwork backpropagation procedure. For the analysis of `LocalLinear` implementations, we assume equally sized input and output sheets, each containing $N = HWC$ neurons, with each output neuron receiving $K = k_h k_w C$ inputs. We further fix the channel count to $C = 1$, the batch size to $B = 96$, and kernel size to $k_h = k_w = 15$, as these are the common settings in our models.

We compare three implementations of the `LocalLinear` layer:

- **Masked linear:** Treats the layer as a fully-connected layer whose connections outside of the local receptive field are masked to zero. Since each of the N output neurons is connected to each of the N input neurons, the weight matrix requires storing N^2 scalars.

- **Pure PyTorch:** Stores the weights efficiently in a tensor of NK elements. However, to obtain the required input-weight products, the internal `unfold` and `index_select` operations produce a tensor that, after materialization into contiguous memory, contains NKB scalars. The total memory required for the layer therefore is $NK(B + 1)$.

- **CUDA implementation (ours):** Stores only the compact weight tensor of NK elements and computes the required products directly in the kernel, thereby requiring substantially less memory.

This difference in memory requirement is illustrated in Figure 5.14, where all other parameters are fixed as above and only the width is varied.

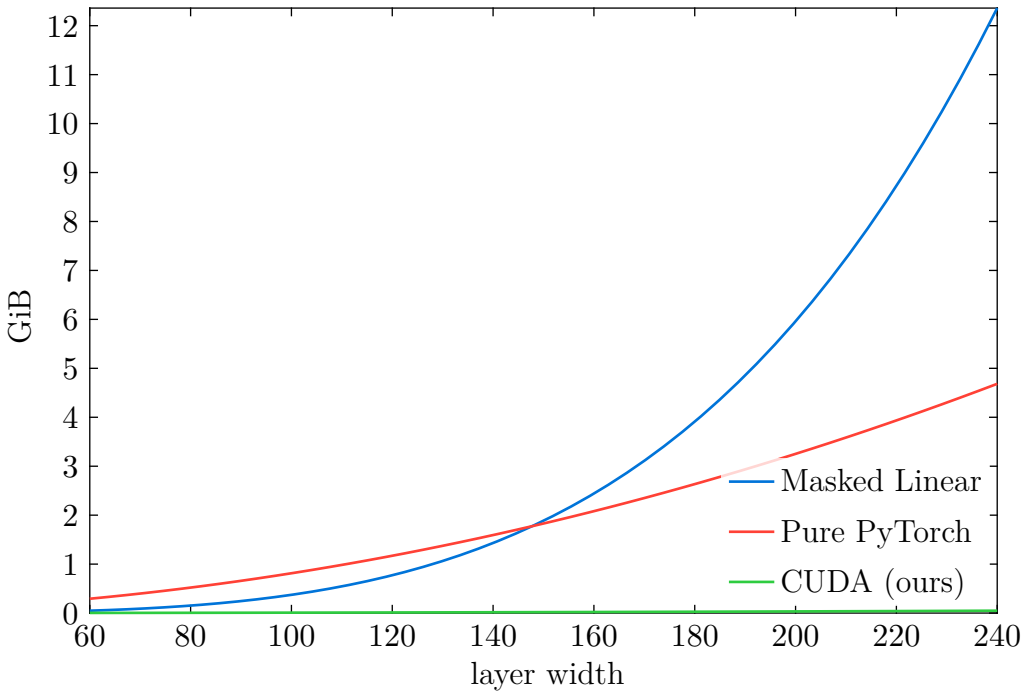


Figure 5.14: Memory requirements of different implementations of the `LocalLinear` layer. The comparison assumes the kernel of size $k_h = k_w = 15$, a single channel in both sheets, batch size $B = 96$, and varying layer width along the x-axis.

To analyze the memory savings introduced by the the two-step backpropagation procedure, we additionally define the chunk size M, which specifies how many weights are calculated at once during a single hypernetwork evaluation. We consider $M = 4\,000\,000$, which is the most commonly used value in our models, as well as $M = 1\,000\,000$, to further illustrate the potential memory savings.

We compare a naive implementation, in which the full computational graph of the hypernetwork used to generate the current `LocalLeinar` weights is retained in memory throughout the computation, with our two-step procedure.

- **Naive implementation:** Requires storing the inputs to every hidden layer of the hypernetwork. To generate all NK weights, this requires $NK \sum_{i=0}^{D_{\mathcal{H}}} L_i$ scalars, where L_i is the input dimensionality of the i-th layer of the hypernetwork and $D_{\mathcal{H}}$ is the depth of the hypernetwork. For a model with D hypernetwork-parameterized layers, the total memory therefore scales as $DNK \sum_{i=0}^{D_{\mathcal{H}}}$.

- **Two-step procedure (ours):** Discards the computational graph immediately after the weights are generated, so only one chunk of weights needs to be represented in memory at a time. Because the chunk size is limited to M, the memory requirement becomes $\min(NK, M) \sum_{i=0}^{D_{\mathcal{H}}}$. This bound no longer scales with the depth D of the model.

Because our method stores only one chunk at a time, the resulting memory savings grow with model depth. This comparison is illustrated in Figure 5.15 for two different chunk sizes, alongside the naive implementation.

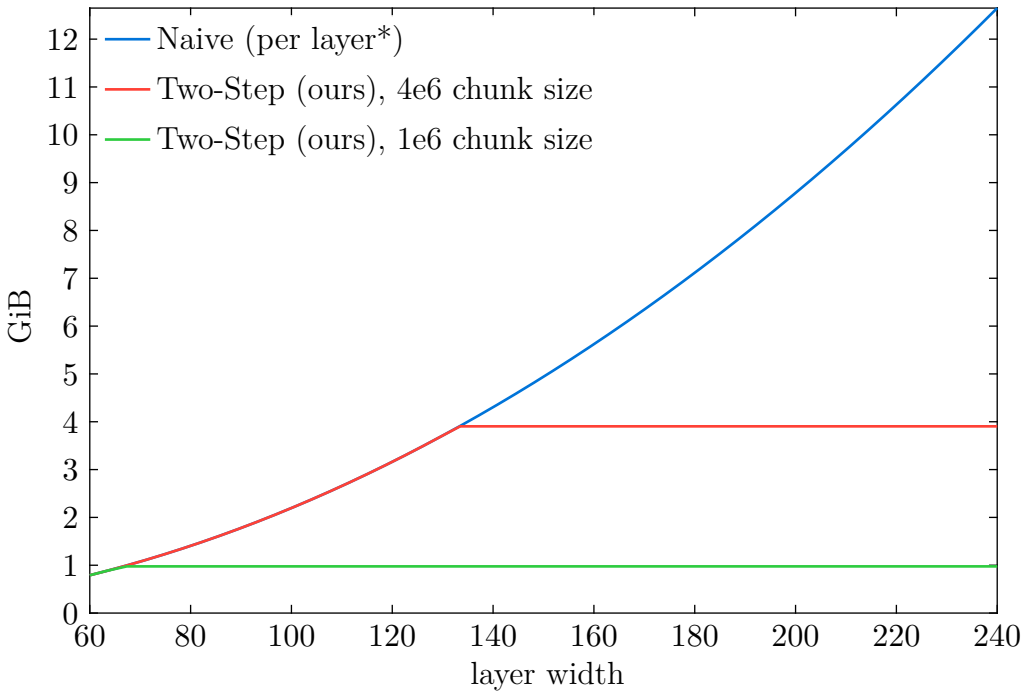


Figure 5.15: Memory requirements of different backpropagation strategies for the hypernetwork parameterization. The comparison assumes the kernel of size $k_h = k_w = 15$, a single channel in both sheets, and varying layer width along the x-axis. The naive implementation stores the hypernetwork computational graph separately for each layer, whereas two-step procedure requires memory only for a single chunk at a time.

6 Discussion

In this thesis, we introduced the `LocalLinear` layer, a generalization of convolution that removes rigid weight sharing and allows mappings between neuronal sheets of arbitrary size and density. This provides a framework for studying biologically inspired deep learning architectures in which features are organized spatially across neuronal sheets rather than across discrete channel dimensions.

Building on this layer, we developed a sequence of parameterizations that progressively improved both local connectivity structure and global topographic organization. The plain `LL` model produced largely unstructured, salt-and-pepper connection fields and highly noisy orientation preference maps. Hypernetwork parameterization substantially improved the structure of local connection fields, especially in the first layer, but produced only weak large-scale organization. Introducing sinusoidal activations into the weight-generating hypernetwork further improved local structure across deeper layers and led to much clearer topographic organization. Finally, parameterizing the neuronal descriptor sheets themselves with SIREN yielded the smoothest and most coherent global organization across the network.

These results suggest that structured local connectivity and smooth population-level organization can emerge from architectural constraints alone, without imposing convolutional weight sharing directly. In particular, the results support a clear progression in representational quality across the model family: `LL` $<$ `LL-Hypr` $<$ `LL-HyprSiren` $<$ `LL-FullSiren` in terms of both connection-field structure and global topographic organization.

A crucial practical result of the thesis is that the framework becomes viable only through the implementation optimizations introduced here. Both the custom CUDA kernel for `LocalLinear` layers and the two-step hypernetwork backpropagation procedure reduce memory usage by orders of magnitude. Without these improvements, the larger models studied here would not fit into the memory available on consumer-grade hardware.

The proposed parameterization also changes the scaling behavior of the model in an important way. In standard convolutional layers, the number of trainable parameters grows with kernel area and with the numbers of input and output channels. In contrast, once both connectivity and descriptor sheets are parameterized continuously, the parameter count of the `LL-FullSiren` model depends primarily on network depth and on the size of the generating networks, rather than on kernel size, sheet density, or channel count. This makes the framework conceptually distinct from standard convolution not only in biological organization, but also in how representational complexity is allocated.

More generally, the framework is also interesting from a representational perspective. Once both the weight-generating hypernetworks and the neuronal descriptor sheets are parameterized continuously, the model is no longer tied to a fixed discrete neuronal grid in the same way as standard convolutional architectures. Instead, both connectivity and descriptor structure can be interpreted as continuous functions over neuronal space, instantiated on a particular sampling grid during execution. Although this possibility was not explored experimentally in the present thesis, it suggests a direction toward models that can in principle be evaluated at different neuronal densities or adapted to non-standard sensor layouts.

From the perspective of computational neuroscience, the present framework moves artificial vision models closer to the spatial organization observed in biological visual systems. Instead of representing features primarily through discrete channel dimensions with rigid shared weights, the proposed models represent them across neuronal sheets with localized, neuron-specific connectivity. This makes it possible to inspect both the connection fields of individual neurons and their population-level organization directly. In this sense, the framework is not only a new deep learning architecture, but also a more interpretable model for studying how structured local connectivity and smooth topographic organization may emerge under biologically inspired constraints.

Beyond its biological motivation, the framework may also be relevant in settings where explicit weight sharing is inconvenient or undesirable, such as neuromorphic hardware and other non–Von Neumann substrates. In such settings, it provides a way to study how structured learning can be achieved without relying on rigid shared weights. More broadly, relaxing exact weight sharing may increase representational capacity for a fixed spatial footprint, which could in principle enable stronger performance on smaller or more constrained hardware. At the same time, the continuous parameterization of connectivity may make it possible to instantiate such structured representations on hardware-specific layouts.

At the same time, several limitations remain. First, although the proposed models achieve promising image-classification performance and can match simpler convolutional baselines, it is not yet clear how the framework should be scaled to substantially larger architectures. Second, the optimization problem is considerably less constrained than in standard convolutional networks, since the architecture introduces many additional hyperparameters related to sheet sizes, kernel radii, descriptor dimensionalities, learning-rate scaling, and SIREN parameterization. Third, while the present thesis demonstrated emergent topographic organization of orientation preference, the resulting maps remain imperfect and do not yet fully reproduce the regularity of biological maps.

Several directions for future work follow directly from the present results:

- **Improving map regularity:** The imperfect regularity of the learned orientation maps suggests that additional mechanisms may be needed. In particular, we suggest three possible directions for future investigation.

    1. Increasing neuronal density, especially in early layers, might be necessary for the model to represent richer local structure and thereby require the smoother locally clustered organization observed in biological networks.

    2. Recurrent lateral interactions may play an important role in the development of biologically realistic orientation preference maps. One example is the model proposed by Antolík (2017).

    3. Regularization, such as distance-weighted L1 or L2 penalties simulating the cost of neuronal wiring, might encourage improved organization.

- **Scaling:** A natural next step is to study scaling more systematically, especially to determine how the framework can be enlarged without losing classification performance or topographic structure.

- **Continuous evaluation:** The continuous parameterization of connectivity and descriptor sheets should be tested more directly by evaluating the model at different neuronal densities or on irregular sensor arrangements. This will likely require mechanisms ensuring that the weights are scaled correctly with density.

Overall, these directions build on the central finding of the thesis: biologically inspired architectural constraints can produce structured connection fields and smooth topographic organization without directly imposing such organization through an auxiliary loss. The proposed framework therefore provides a starting point for further work on trainable, interpretable, and spatially organized deep visual models.

Bibliography

Afgoustidis, A. (2015). Orientation Maps in V1 and Non-Euclidean Geometry. *Journal of Mathematical Neuroscience*, *5*, 24. https://doi.org/10.1186/s13408-015-0024-7

Antolík, J. (2017). Rapid Long-Range Disynaptic Inhibition Explains the Formation of Cortical Orientation Maps. *Frontiers in Neural Circuits*. https://doi.org/10.3389/fncir.2017.00021

Bear, M. F., Connors, B. W., & Paradiso, M. A. (2016). *Neuroscience: Exploring the Brain* (4rd ed). Jones & Bartlett Learning.

Bosking, W. H., Zhang, Y., Schofield, B., & Fitzpatrick, D. (1997). Orientation Selectivity and the Arrangement of Horizontal Connections in Tree Shrew Striate Cortex. *The Journal of Neuroscience*, *17*(6), 2112–2127. https://doi.org/10.1523/JNEUROSCI.17-06-02112.1997

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018,). *JAX: composable transformations of Python+NumPy programs*. http://github.com/jax-ml/jax

Coubard, O. A., Urbanski, M., Bourlon, C., & Gaumet, M. (2014). Educating the blind brain: a panorama of neural bases of vision and of training programs in organic neurovisual deficits. *Frontiers in Integrative Neuroscience*. https://doi.org/10.3389/fnint.2014.00089

Daugman, J. G. (1985). Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *J. Opt. Soc. Am. A*, *2*(7), 1160–1169. https://doi.org/10.1364/JOSAA.2.001160

Dayan, P., & Abbott, L. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems* (Vol. 15).

Derrington, A. (2001). The lateral geniculate nucleus. *Current Biology*, *11*(16), R635–R637. https://doi.org/10.1016/S0960-9822(01)00379-7

DiCarlo, J. J., Zoccolan, D., & Rust, N. C. (2012). How Does the Brain Solve Visual Object Recognition?. *Neuron*, *73*(3), 415–434. https://doi.org/10.1016/j.neuron.2012.01.010

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *Corr*. https://arxiv.org/abs/2010.11929

Eccles, J. C., Fatt, P., & Koketsu, K. (1954). Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurones. *The Journal of Physiology*, *126*(3), 524–562. https://doi.org/10.1113/jphysiol.1954.sp005226

Eldan, R., & Shamir, O. (2016,). *The Power of Depth for Feedforward Neural Networks*. https://arxiv.org/abs/1512.03965

Felleman, D. J., & Van Essen, D. C. (1991). Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral Cortex*, *1*(1), 1–47. https://doi.org/10.1093/cercor/1.1.1-a

Freeman, J., Ziemba, C. M., Heeger, D. J., Simoncelli, E. P., & Movshon, J. A. (2013). A functional and perceptual signature of the second visual area

in primates. *Nature Neuroscience*, *16*(7), 974–981. https://doi.org/10.1038/nn.3402

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. https://doi.org/10.1007/BF00344251

Geirhos, R., Janssen, D. H. J., Schütt, H. H., Rauber, J., Bethge, M., & Wichmann, F. A. (2017). Comparing deep neural networks against humans: object recognition when the signal gets weaker. *Corr*. http://arxiv.org/abs/1706.06969

Gerstner, W., & Kistler, W. (2002). *Spiking Neuron Models* (p.). https://doi.org/10.1017/CBO9780511815706.002

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Ha, D., Dai, A., & Le, Q. V. (2016, December). *HyperNetworks*. arXiv. https://doi.org/10.48550/arXiv.1609.09106

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-Inspired Artificial Intelligence. *Neuron*, *95*(2), 245–258. https://doi.org/10.1016/j.neuron.2017.06.011

He, K., Zhang, X., Ren, S., & Sun, J. (2015b, February). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. arXiv. https://doi.org/10.48550/arXiv.1502.01852

He, K., Zhang, X., Ren, S., & Sun, J. (2015a, December). *Deep Residual Learning for Image Recognition*. arXiv. https://doi.org/10.48550/arXiv.1512.03385

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544. https://doi.org/https://doi.org/10.1113/jphysiol.1952.sp004764

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, *160*(1), 106–154. https://doi.org/10.1113/jphysiol.1962.sp006837

Jones, J., & Palmer, L. A. (1987). An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 1233–1258. https://api.semanticscholar.org/CorpusID:16809045

Kaschube, M., Schnabel, M., Löwel, S., Coppola, D. M., White, L. E., & Wolf, F. (2010). Universality in the Evolution of Orientation Columns in the Visual Cortex. *Science*, *330*(6007), 1113–1116. https://doi.org/10.1126/science.1194869

Kingma, D. P., & Ba, J. (2017,). *Adam: A Method for Stochastic Optimization*. https://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Lapicque, L. (1907). Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *J. Physiol. Pathol. Gen.*, *9*, 620–635.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

Lee, H., & DiCarlo, J. J. (2018,). Topographic Deep Artificial Neural Networks (TDANNs) predict face selectivity topography in primate inferior temporal (IT) cortex. *2018 Conference on Cognitive Computational Neuroscience*. https://doi.org/10.32470/CCN.2018.1085-0

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861–867. https://doi.org/https://doi.org/10.1016/S0893-6080(05)80131-5

Liao, Q., & Poggio, T. (2020, December). *Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex*. arXiv. https://doi.org/10.48550/arXiv.1604.03640

Lu, Z., Doerig, A., Bosch, V., Krahmer, B., Kaiser, D., Cichy, R. M., & Kietzmann, T. C. (2025). End-to-end topographic networks as models of cortical map formation and human visual behaviour. *Nature Human Behaviour*, *9*(9), 1975–1991. https://doi.org/10.1038/s41562-025-02220-7

Martín~Abadi, Ashish~Agarwal, Paul~Barham, Eugene~Brevdo, Zhifeng~Chen, Craig~Citro, Greg~S.~Corrado, Andy~Davis, Jeffrey~Dean, Matthieu~Devin, Sanjay~Ghemawat, Ian~Goodfellow, Andrew~Harp, Geoffrey~Irving, Michael~Isard, Jia, Y., Rafal~Jozefowicz, Lukasz~Kaiser, Manjunath~Kudlur, … Xiaoqiang~Zheng. (2015,). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/

Masland, R. H. (2012). The Neuronal Organization of the Retina. *Neuron*, *76*(2), 266–280. https://doi.org/10.1016/j.neuron.2012.10.002

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Ohki, K., Chung, S., Kara, P., Hübener, M., Bonhoeffer, T., & Reid, R. C. (2006). Highly ordered arrangement of single neurons in orientation pinwheels. *Nature*, *442*, 925–928. https://doi.org/10.1038/nature05019

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). *Automatic differentiation in PyTorch*.

Pogodin, R., Mehta, Y., Lillicrap, T. P., & Latham, P. E. (2022, January). *Towards Biologically Plausible Convolutional Networks*. arXiv. https://doi.org/10.48550/arXiv.2106.13031

Poli, M., Dupoux, E., & Riad, R. (2022,). *Introducing topography in convolutional neural networks*. https://arxiv.org/abs/2211.13152

Prince, S. J. (2023). *Understanding Deep Learning*. The MIT Press. http://udlbook.com/

Simonyan, K., & Zisserman, A. (2015, April). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv. https://doi.org/10.48550/arXiv.1409.1556

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., & Wetzstein, G. (2020, June). *Implicit Neural Representations with Periodic Activation Functions*. arXiv. https://doi.org/10.48550/arXiv.2006.09661

Straka, M. (2026,). *NPFL138: Deep Learning — Lecture 04: Convolutional Neural Networks*. https://ufal.mff.cuni.cz/~straka/courses/npfl138/2526/slides/?04#12

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014, September). *Going Deeper with Convolutions*. arXiv. https://doi.org/10.48550/arXiv.1409.4842

Vančura, V., Kordík, P., & Straka, M. (2024). beeFormer: Bridging the Gap Between Semantic and Interaction Similarity in Recommender Systems. *18th ACM Conference on Recommender Systems*, 1102–1107. https://doi.org/10.1145/3640457.3691707

Wu, J., Zhang, Q., & Xu, G. (2017,). *Tiny ImageNet Challenge*. https://api.semanticscholar.org/CorpusID:212697711

Yamins, D. L. K., & DiCarlo, J. J. (2016). Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, *19*(3), 356–365. https://doi.org/10.1038/nn.4244

List of Figures

1.1 The basic parts of a neuron. . . . 10
1.2 The internal structure of a typical neuron. . . . 11
1.3 The uneven distribution of ions across the neuronal membrane, giving rise to the membrane potential. . . . 12
1.4 An action potential displayed by an oscilloscope. . . . 13
1.5 A spike train recording. . . . 13
1.6 The axon terminal and the synapse. . . . 14
1.7 The eye in cross section. . . . 15
1.8 Retinofugal projection. . . . 16
1.9 LGN organization. . . . 16
1.10 Visual areas in the human brain. . . . 17
1.11 The retinotopic map in the striate cortex. . . . 17
1.12 The basic system of retinal information processing. . . . 18
1.13 Direct and indirect pathways from photoreceptors to bipolar cells. . . . 19
1.14 A center-surround ganglion cell receptive field. . . . 19
1.15 The organization of the LGN. . . . 20
1.16 The lamination of the striate cortex. . . . 21
1.17 Orientation selectivity. . . . 22
1.18 Gabor filter examples illustrating different angles and phases. . . . 22
1.19 A complex cell receptive field. . . . 23
1.20 An orientation preference map observed in the visual cortex of a tree shrew. . . . 23
1.21 Extrastriate cortex in the macaque monkey brain. . . . 24

2.1 Artificial neuron model. . . . 25
2.2 Fully-connected feedforward neural network with one hidden layer and two outputs. . . . 26
2.3 A computational graph for the neural network defined in 29
2.4 A 2D convolutional layer utilizing multiple different kernels to extract features and stack them onto separate output channels. . . . 31
2.5 ResNet residual block. . . . 31
2.6 Example network architecture of ResNet-34. . . . 32
2.7 A hypernetwork generates the weights for a feedforward network. . . . 33

4.1 Finding $\phi(i)$ and $\phi'(j)$ in a `LocalLinear` layer. . . . 36
4.2 Comparing valid and same padding in `LocalLinear` layers. . . . 37
4.3 Architecture of the weight-generating hypernetwork. . . . 39
4.4 Indices used in the hypernetwork formulation. . . . 40
4.5 Architecture of the SIREN-based weight-generating hypernetwork. . . . 42
4.6 Architecture of the SIREN-based neuronal-descriptor network. . . . 42
4.7 The model used in the main experiments, utilizing the `LocalLinear` layers. . . . 44
4.8 Neuronal descriptor sheets assigned to the input and 44
4.9 The convolutional baseline 45
4.10 Example sinusoidal gratings used to estimate orientation preference maps. . . . 47
4.11 Example visualizations. . . . 47

5.1 Connection fields of the `LL` model. . . . 49

5.2 Orientation preference map of the `LL` model. 50
5.3 Neuronal descriptor sheet of the `LL-Hypr` model. 50
5.4 Orientation preference map of the `LL-Hypr` model. 50
5.5 Connection fields of the `LL-Hypr` model. 51
5.6 Connection fields of the `LL-HyprSiren` model. 52
5.7 Neuronal descriptor sheet of the `LL-HyprSiren` model. 52
5.8 Orientation preference map of the `LL-HyprSiren` model. 53
5.9 Neuronal descriptor sheet of the `LL-FullSiren` model. 53
5.10 Orientation preference map of the `LL-FullSiren` model. 54
5.11 Connection fields of the `LL-AllSiren` model. 54
5.12 Training performance of the `LL-HyprSiren` model compared with the convolutional baselines. 55
5.13 Training performance of all proposed `LocalLinear` models with different parameterizations. 56
5.14 Memory requirements of different implementations of the `LocalLinear` layer. 58
5.15 Memory requirements of different backpropagation strategies for the hypernetwork parameterization. 59

7.1 Neuronal descriptor sheet with learning-rate scale 100. 70

List of Tables

4.1 Comparison of the proposed architecture with the two convolutional baselines. 46
4.2 The parameter counts and FLOPs for the `LocalLinear` models as well as convolutional baselines. 46
4.3 Hyperparameters used for training. 48

5.1 Summary of image classification performance, trainable parameter counts, and FLOPs for all compared models. 56

7.1 Preliminary comparison of dropout values across architecture setups.. 70
7.2 Preliminary comparison of neuronal descriptor learning-rate scales. . 70

A Preliminary Hyperparameter Selection

The following tables report preliminary comparisons used to select two training hyperparameters: dropout and neuronal descriptor learning-rate scaling. These experiments were not intended as exhaustive ablation studies, but rather as empirical justification for the values used in the main experiments. Each entry reports the best validation accuracy obtained during 120 training epochs. The setup index denotes a preliminary `LL-HyprSiren` architecture variant.

Table 7.1 illustrates the importance of dropout for performance.

Parameter	*Setup*				
	1	**2**	**3**	**4**	**5**
Dropout = 0.0	12.4%	11.7%	10%	9.7%	11.7%
Dropout = 0.2	15.4%	15.2%	11.7%	13.2%	14.3%

Table 7.1: Preliminary comparison of dropout values across architecture setups. Each entry reports the best validation accuracy obtained during 120 training epochs. Green highlights indicate the stronger dropout choice for each setup.

Table 7.2 shows that a larger learning-rate scale for neuronal descriptors improves performance, and Figure 7.1 illustrates how a low learning-rate scale leads to poor organization of the neuronal descriptor sheets.

Parameter	*Setup*			
	1	**2**	**3**	**4**
Neuronal descriptor LR scale = 100	10.6%	9.9%	10.8%	11.3%
Neuronal descriptor LR scale = 300	10.6%	11.8%	11.5%	12%
Neuronal descriptor LR scale = 1000	11.7%	12.6%	12.5%	13%

Table 7.2: Preliminary comparison of neuronal descriptor learning-rate scales. Entries report the best validation accuracy over 120 training epochs. Green highlights indicate the strongest value for each setup.

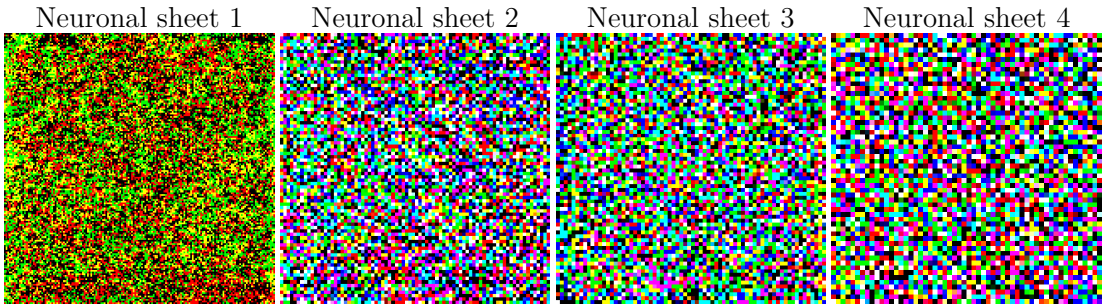


Figure 7.1: Neuronal descriptor sheet with learning-rate scale 100. At this low scale, the descriptor sheet remains close to its initial random pattern, indicating insufficient adaptation during training.

B Attachments

The electronic attachment to this thesis is provided as a `.tar.gz` archive containing the source code used for the experiments. It includes the implemented library `biotorchvision`, experiment execution scripts, configuration files, and custom CUDA kernels.

The main files and directories are:

- `README.md`: instructions for setting up the Python environment, compiling the CUDA extensions, and running the experiments.
- `scripts/cnn_run.py`: main script for launching experiments.
- `scripts/architectures/universal.py`: definition of the architecture using `LocalLinear` layers and hypernetwork parameterizations.
- `configs/presets/`: experiment configuration presets. This directory includes the configurations used for the thesis experiments as well as additional exploratory configurations not reported in the thesis.
- `biotorchvision/`: library containing the implemented layers, parameterizations, visualization tools, and supporting utilities.
- `biotorchvision/layers/_local_linear_2d.py`: implementation of the `LocalLinear` layer.
- `biotorchvision/ops/`: CUDA kernels and PyTorch interface for the custom operations.
- `biotorchvision/layers/parametrizations/hyprweights.py`: hypernetwork parameterization of generated weights.
- `biotorchvision/layers/parametrizations/siren_sheet.py`: SIREN-based implicit neural representation for neuronal descriptor sheets.