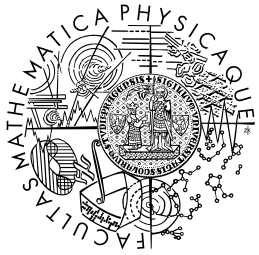

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Martin Picek

Rotation-equivariant convolutional neural network for design of visual prosthetic stimulation protocol

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: Mgr. Ján Antolík, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

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

First and foremost, I want to thank my supervisors. With excellent teaching skills and a friendly attitude, Mgr. Ján Antolík, Ph.D. inspired me and many other students to work in the field of computational neuroscience. I was fortunate to have such a patient supervisor willing to devote a great deal of time to his students. I am grateful for all the invaluable knowledge that I was given from him.

Luca Baroni was the person to whom I could come for any help. Despite his very tight schedule, he offered me every spare minute of his time to support me in writing this thesis. I am glad that during this journey, we became not only good colleagues but, most importantly, very good friends.

Furthermore, I want to acknowledge Dávid, David, Tomáš, and Radovan, who supported me throughout writing this thesis. Thank you, Vendulka, MzM, and other friends, for giving me all the emotional care I needed.

And most importantly, I want to thank my family, for the never-ending love they give me.

Title: Rotation-equivariant convolutional neural network for design of visual prosthetic stimulation protocol

Author: Martin Picek

Department of Software and Computer Science Education: Department of Software and Computer Science Education

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

Abstract: Neighboring neurons in the primary visual cortex (V1), the first cortical area processing visual information, are selective to stimuli presented in neighboring positions of the visual field with a specific edge orientation. In this way, they form the so-called retinotopic and orientation maps of V1. Due to the absence of high-resolution cortical stimulation devices, vision restoration through prosthetic implants in V1 has not yet taken advantage of the orientation maps. However, the availability of cortical implants with stimulation resolution high enough to target separate orientation columns can be anticipated soon.

Since other stimulus features are also encoded in the cortex, such as color, size, or phase, but cannot be reliably engaged even by high-resolution stimulation, in this thesis, we ask the question of how well can visual stimuli be encoded in V1 if only orientation and position preference is known. To address this question, we propose a deep neural network (DNN) providing a scalar neural activity descriptor for any targeted cortical location and multiple different orientations. This is achieved by employing a rotation-equivariant convolutional neural network (reCNN) with the last layer having only one channel for each orientation, returning the desired three-dimensional feature tensor. A specialized readout estimates the neurons' positions and their orientation preference, using them to yield one value per neuron from the core's last layer, from which the neural response is predicted. These scalar features might serve as an input for the future stimulation protocol.

The proposed network was trained both on an experimental and a synthetic dataset. The synthetic dataset generated by Antolík et al. model of a cat's V1 provided locations of neurons along with their orientation preference. This was used to examine the DNN model's precision in both neural response prediction and neural location and preferred orientation estimates.

This work proposes a DNN model yielding a 0.8327 correlation fraction with respect to the control model. Moreover, with the proposed reCNN model achieving sufficient performance, we were able to reconstruct the in-silico orientation maps. This promising result suggests that targeting cortical columns of neurons matching in orientation and position preference might improve information encoding via prosthetic stimulation.

Keywords: deep neural networks, computational neuroscience, rotation-equivariant CNN, convolutional neural network, cortical prosthetics, stimulation protocol

Contents

Introduction

Despite the effort to eradicate it, blindness is still a common condition that worsens the quality of people's lives. A recent study showed that in 2015, there were an estimated 36 million blind people [Ackland et al., 2017]. Although a cure has been found for some diseases linked to blindness, some causes of blindness are still incurable.

Among many different approaches to restoring sight to the blind, one possible solution is brain implants [Kilgore, 2015]. This thesis focuses on visual prostheses implanted into the primary visual cortex (also called V1 or the striate cortex). However, visual neural prostheses can also target other stages of the early visual pathway. They are sometimes implanted into the retina or the lateral geniculate nucleus (LGN), depending on which part of the pathway remains viable in the given subject [Mirochnik and Pezaris, 2019].

The first intracortical pioneering models of V1 targeting sensory prosthetics are currently being clinically tested [Lewis and Rosenfeld, 2016] [Fernández et al., 2021]. Nonetheless, these devices do not provide a high-resolution sight yet. However, better stimulation technology and stimulation protocols are in development. In this work, the latter will be the subject of our attention.

As of yet, V1 neural prosthetic implants have taken advantage of only one important characteristic of the spatial organization of the human V1 brain area - retinotopic mapping - a mapping from the subject's visual field to V1 preserving the topological structure of the visual field (more in Subsection 1.3.1). However, another significant characteristic has not been exploited in the prosthetic protocols yet - the orientation selectivity of V1 cortical neurons, which was discovered by Hubel and Wiesel in 1962 [Hubel and Wiesel, 1962] paving them the way to a Nobel prize in 1981. Furthermore, V1 neurons form the so-called orientation maps: neighboring neurons have very similar orientation preferences with relatively slow and continuous preference changes in space.

Although this important feature of the striate cortex has been known for many years, there were no high-resolution stimulation techniques that would allow for targeting cortical orientation columns separately. However, new devices are on the way to achieving resolution of prosthetic stimulation matching the orientation columns, which presents an opportunity to stimulate the V1 neurons according to their orientation domain, preserving even more encoding information properties (along with retinotopy) and ideally leading to better sight restoration.

It should be noted that retrieving cortical orientation maps in blind subjects is still an open problem. The solution might, however, come soon from the research performed on sighted non-human mammals [Smith et al., 2018], where the reconstruction of orientation maps was shown to be possible by analyzing patterns of spontaneous activity. Validation of the method to identify orientation domains in humans has still to be performed. Nonetheless, research in this area continues, and it is likely that soon there will be techniques to identify orientation domains in people with acquired blindness. Combined with the high-resolution stimulation devices, this would open the path to the utilization of the orientation selectivity encoding in evoking visual perception in V1.

In this thesis, we build upon this assumption and propose a stimulation pro-

tocol that takes advantage of the knowledge of preferred orientation along with a device with a resolution high enough to target cortical columns separately.

Goals

The underlying hypothesis of the present work is that the representation of the visual stimulus in the cortical population activity determines the perception. Under this assumption, in order to stimulate the primary visual cortex to elicit a visual percept as close as possible to the input image, we need to know how neurons would respond to a given input image naturally. Subsequently, the stimulation device will try to evoke the same activity pattern to recreate the original percept. Thus, understanding the encoding properties of neurons targeted by the prosthetic implant is essential for successfully implementing cortical prosthetic stimulation protocol for sensory restoration.

In a blind patient, precise knowledge of the visual stimuli encoding under the implant is impossible, as the interrupted visual pathway between the eye and cortex does not allow probing the selectivity of the V1 neurons. In this thesis, we will instead assume that our knowledge of encoding under the implant is restricted to the retinotopic and orientation selectivity of the targeted neurons.

It is, however, an open question how precise activity patterns can be elicited based only on the knowledge of the encoding of orientation and retinotopy. This work is the first to investigate this problem. We aim to estimate the lower bound of the achievable neural response prediction performance given an input image, positions of receptive fields, and preferred orientations of the population of targeted neurons. This result can serve as the first rough estimate of the achievable quality of a stimulation protocol taking into account retinotopy and orientation preferences of the cortical tissue. Moreover, we create a data-driven model, producing a feature map with a scalar value for every position and preferred orientation. This feature map can be used for the design of a stimulation protocol taking into account the preferred orientation along with retinotopy. Implementation of the prosthetic protocol for a particular stimulation device is, however, beyond the scope of this thesis since it depends on many other factors, mainly on the implant itself.

To estimate the mentioned lower bound, we developed a neural response predicting deep neural network model (DNN) designed explicitly for this task. Such DNN is composed of a rotation-equivariant core and a specialized readout. The core's purpose is to process input images extracting features in all possible orientations with the last layer constrained to have only one channel. Due to the core's equivariant properties, such a bottleneck enforces the units of the core's output to be generated by only one specific computation on the input image but applied in all positions and for different orientations. Based on the data, the readout associates a specific position and orientation for each neuron and simultaneously learns to predict responses from the unit of the core's output corresponding to the learned position and orientation of a neuron in question.

The model's architecture is inspired by the idea presented in the work of Lurz et al. [Lurz et al., 2021] where most of the computation is moved into the core, and its ability to generalize between different subjects is investigated afterward. Similarly, our work aims to develop a core performing a general computation

provided by the striate cortex, with a readout accounting for only the subject's individual differences.

Compared to Lurz and his colleagues, we do not possess stimulus-response pairs recorded on different subjects of the same species. Therefore, we cannot directly assess the generalization property of our core. However, the readout fetches a scalar value from a certain position from the core's bottleneck and learns only a linear function for each neuron to map the selected value onto the neural response. With this approach, only two parameters per neuron need to be fitted, disregarding the features for position and preferred orientation estimation. As a consequence, the readout's computational capacity devoted to neural response prediction is strictly limited. With this approach, we train a core to perform almost all the complex computations of the primary visual cortex, providing the desired feature map in the core's last layer with a value for every orientation and position. This feature map could be leveraged in future work for a concrete stimulation of the V1 tissue, provided we know its orientation preferences.

Since we have at our disposal a dataset generated from a computational model of cat's V1 developed by Antolík et al. [Antolík et al., 2019] with neurons' positions and their orientation preference available, we test the accuracy of the estimates of the neurons' positions and their preferred orientations provided by the readout. Moreover, having all the necessary data available, we aim to reconstruct the in-silico model's orientation maps. This represents a novel approach to estimating orientation maps from recordings of a population of neurons' responses to a set of natural images.

1. The Visual System

This chapter covers the neuroscientific context of the thesis. The main part will outline relevant topics concerning the primary visual cortex. For a broader neuroscientific review, we refer the reader to the book Neuroscience: Exploring the brain [Bear et al., 2020].

1.1 The Retina

The complex process of creating a visual percept in our minds begins in the eye. Firstly, the light passes through the eye's optical system and projects on the retina, where the first neural response to the light happens. Transformation of the light into a neural response is accomplished by two types of photoreceptors; rods and cones. Rods do not need a great deal of light to be activated; they are therefore essential for night vision. Cones are complementary, requiring much more light to elicit activity, but their purpose is different; cones provide us with color vision and are smaller, thus providing higher-resolution vision. Three types of cones detect different color spectrum ranges - red, green, and blue.

The visual space projected on the retina and thus perceived by the visual system is called the visual field. It is mainly limited by the eyes' physiological structure and position in the head. Sometimes we are interested in the area of the visual field where the presence of a stimulus influences the neural activity of a particular cell. This area is called a neuron's receptive field. The further the cell is in the visual pathway, the larger receptive field it tends to have since it aggregates information from a greater visual field area. The size of an object in our visual field is measured in degrees of visual angle, which describes the distance the object subtends on the retina. As an example, the Moon spans approximately 0.5 degrees of visual angle. One millimeter of retina corresponds to roughly 3.5 degrees of visual angle [Bear et al., 2020].

After the light is encoded into the neural signal by photoreceptors, the information is sent to bipolar cells and subsequently to ganglion cells. Through horizontal and amacrine cells, ganglion cells obtain aggregated information from the local photoreceptors, allowing them to perform the first nontrivial information processing.

There are two main types of ganglion cells; ON and OFF cells. Their circular receptive field is divided into two regions; a circular center and a surrounding disc. ON and OFF cells are active when they detect a significant luminance difference between the center and the surround. ON cells fire when the center detects proportionally higher luminance than the surround, while OFF cells are active in the opposite situation.

Nevertheless, there are many more types of ganglion cells and other retinal cells, and their functions vary. In conclusion, the retina can detect local changes in contrast or even detect objects' movement.

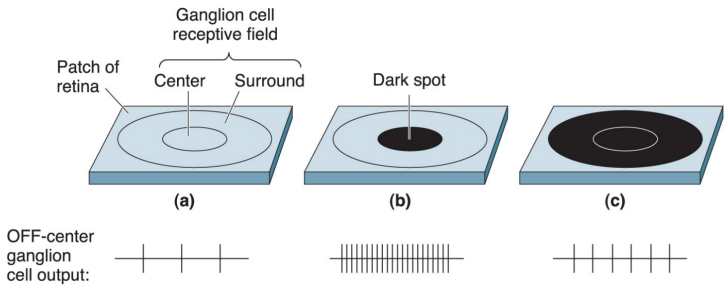

Figure 1.1: Illustration of the OFF cell's response given different light settings. The highest firing rate of the simple cell is produced when a dark spot surrounded by light is present in the OFF cell's receptive field. Adapted from [Bear et al., 2020].

1.2 The Lateral Geniculate Nucleus (LGN)

When the visual information is encoded and partially processed in the ganglion cell activity, it is sent out of the eye into the brain. The neural signal travels via the optical nerve into the Lateral Geniculate Nucleus (LGN), which is a gateway to the primary visual cortex. LGN is an evolutionarily old brain area located in the dorsal thalamus, and it is connected to numerous different parts of the brain, consequently modulating visual perception. For this reason, recent papers that predict the neural responses in V1 take into account information correlated to other sources of input to LGN [Sinz et al., 2018], for example, locomotion, gaze position, or emotional state of the subject. It is vital to understand that visual processing depends not solely on the visual stimulus but also on other additional information. Without this, neural responses in the V1 can never be entirely explained.

Right now, research is not focused on this phenomenon. However, if cortical prostheses were to be used in the future, additional data about the implant user would be necessary to acquire the best prosthetic performance.

Visual percept is further processed in LGN in a similar way as it is accomplished in the retina (but with a wider receptive field) and then forwarded to the V1. We must mention that LGN is not a trivial linear gateway to V1. Most of the LGN's input is paradoxically from the visual cortex. It is unclear what the role of this recurrence is, mainly due to the difficulty of research on this old brain structure lying deep in the middle of the brain, making recording or stimulation experiments targeting this structure challenging.

1.3 The Primary Visual Cortex

1.3.1 The Primary Visual Cortex and Retinotopy

The first cortical area that is reached by the visual information is the primary visual cortex, also referred to as V1. V1 exhibits the characteristic feature of the central visual system; retinotopy. Retinotopy is a mapping of neural signals from neighboring retinal neurons to neighboring neurons of the LGN and the primary

visual cortex. The topology of the visual field is therefore maintained even in the V1.

The striate cortex is about 2 mm thick. It is morphologically divided into six layers, usually numbered with Roman numbers I to VI, I being the outermost layer and VI the innermost. The most significant portion of the input from the LGN is projected onto layer IV, in particular to sublayer IVC. Many intracortical connections are lateral (parallel to the cortical surface), contributing to combining information from the local neighborhood. Further connections are perpendicular to the cortical surface running through other layers up to layer I, feeding the downstream neurons with input from the same retinal area and maintaining the retinotopic spatial organization in the subsequent layers (Figure 1.2).

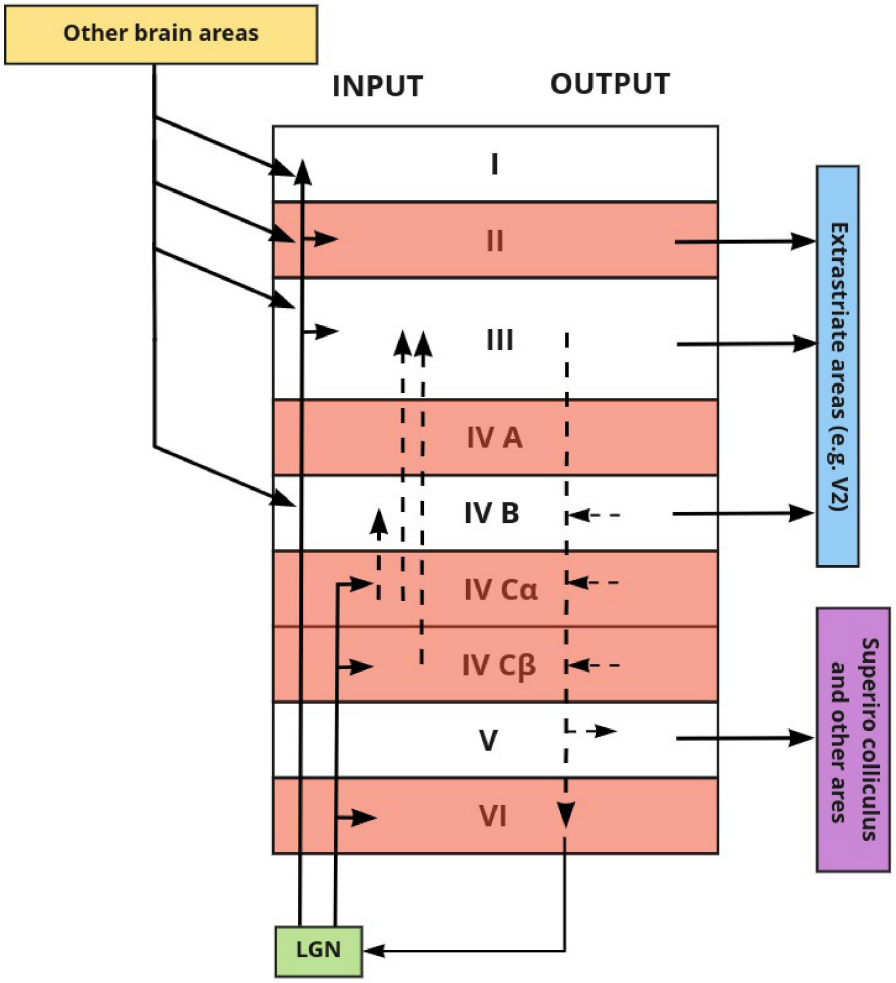

Figure 1.2: A simplified schema of neural input and output of the primary visual cortex. Dashed lines symbolize interconnections between cortical layers. For reasons of clarity, the lateral connections are excluded from this image. Inspired by [Bear et al., 2020] and [Kandel et al., 2000].

1.3.2 Orientation Selectivity

Besides retinotopy, cells in the previous brain areas on the visual pathway, the retina, and the LGN, have circular receptive fields and some cells manage to detect local changes of luminance in a given point and its surroundings. On the other hand, V1 needs more sophisticated visual patterns for the cells to elicit significant activity. In a famous experiment by Hubel and Wiesel in 1962 [Hubel and Wiesel, 1962], V1 neurons were found to produce the most significant neural response to elongated bars of light moving across the cell's receptive field. Furthermore, the cell's response is selective to the orientation of the bar.

Neurons in the primary visual cortex behave very specifically; many prefer one exact stimulus orientation over the other orientations. If we rotate the stimulus, the response continually decreases until the bar is perpendicular to the preferred orientation, yielding a minimal response (Figure 1.3). The function describing the response dependency on the orientation of the bar is called the orientation tuning curve. The narrower the tuning curve is, the more orientation selective the cell is said to be.

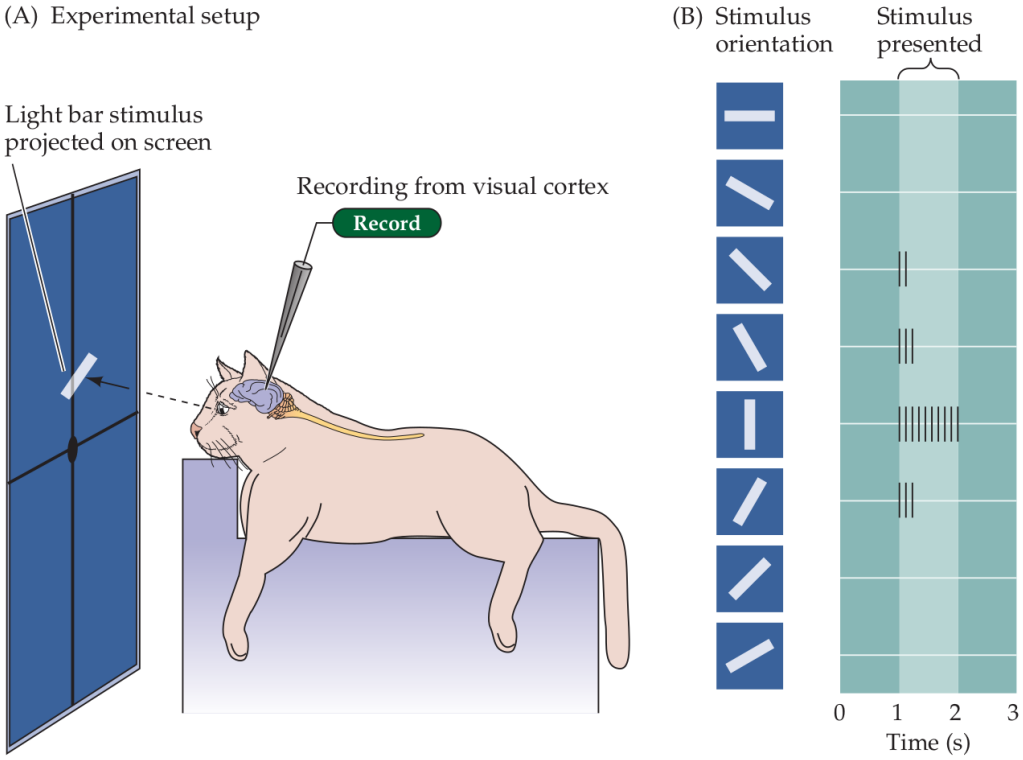

Figure 1.3: Adapted from [Purves, 2019]. (A) An anesthetized cat is equipped with contact lenses to center its gaze on the black dot in the middle of a screen. A white bar of light is presented in the recorded cell's receptive field in different directions. An extracellular electrode is injected into the cat's brain to record the particular neuron in the primary visual cortex. (B) The cell's response is dependent on the orientation of the bar. The closer the angle is to the preferred orientation, the higher frequency of spikes is generated by the neuron.

Orientation preference in the striate cortex also has characteristic spatial properties. As we already know, neurons located on a perpendicular line to the cortical

surface process information from the exact location on the retina and the corresponding location in the visual field. Similarly, these neurons also have the same preferred orientation. Moreover, in higher mammals, neighboring cells in the tangential direction to the cortical surface possess similar orientation preferences, which is not the case in rodents [Van Hooser et al., 2005], [Bednar and Wilson, 2016]. If we inserted an electrode tangentially into the higher-mammalian V1 tissue, we would observe the orientation preference to periodically and continuously change (Figure 1.4).

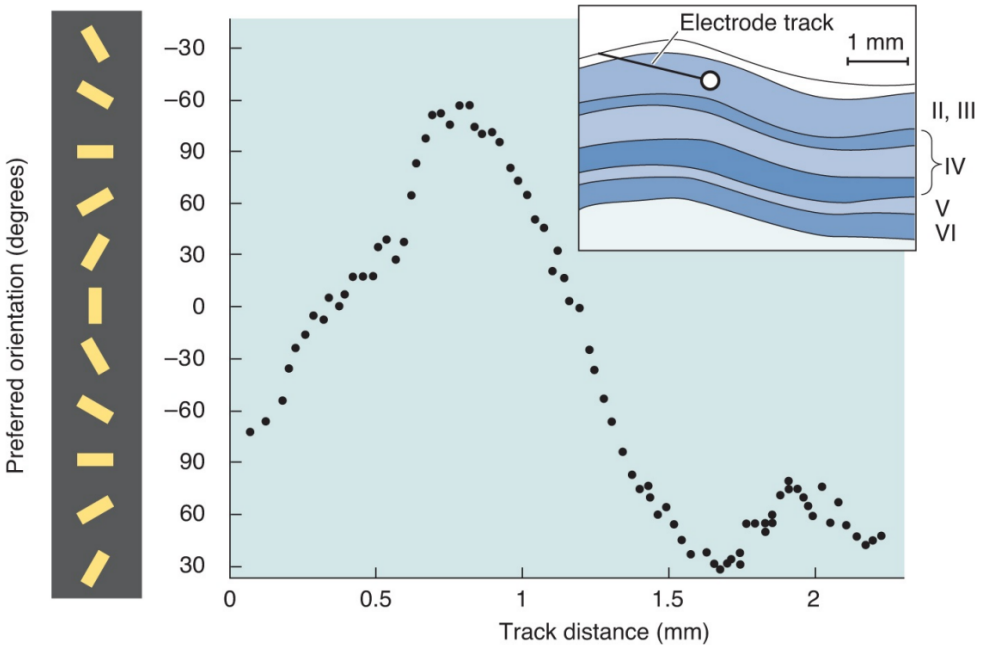

Figure 1.4: Since tangentially neighboring cells have similar orientation preferences, tangental insertion of a recording electrode into the striate cortex of higher mammals records a continuous change of the orientation preference. Moreover, this change is periodic. Adapted from [Bear et al., 2020].

Consequently, the primary visual cortex can be functionally organized into orientation columns, columns of cells parallel to V1, comprising neighboring neurons with very similar orientation selectivity (Figure 1.5).

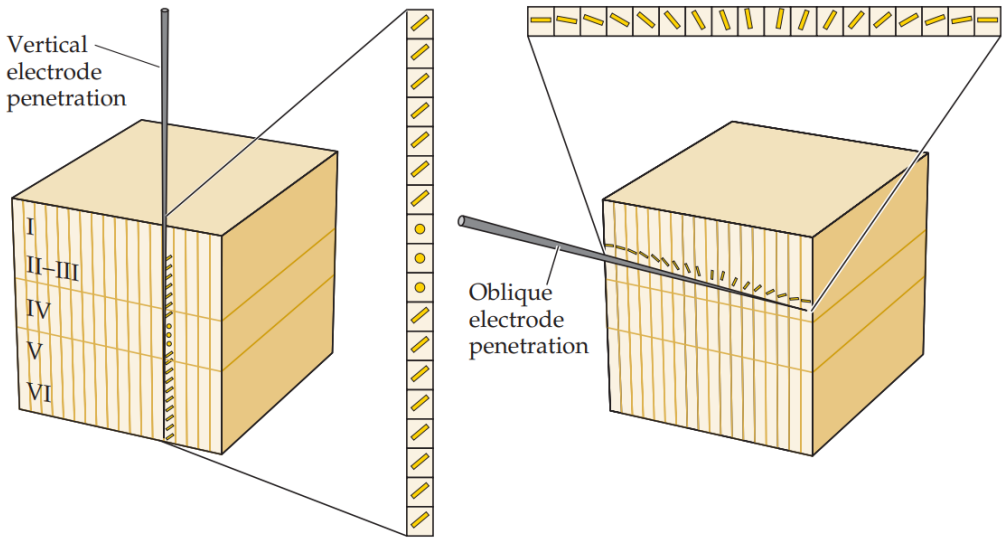

Figure 1.5: Orientation columns in monkey V1. As a measurement from the electrodes suggests, when the electrode is inserted vertically (perpendicularly to the cortex), the preferred orientation does not change. If inserted tangentially (obliquely) to the cortex, the preferred orientation shifts continually and periodically. Adapted from [Purves, 2019].

It is possible to reconstruct and visualize the preferred orientation in the striate cortex [Smith et al., 2018]. The result is called an *orientation map*, and it is essential to note that it differs from subject to subject (Figure 1.6). In higher mammals, the neighboring regions of this map, where the preferred orientations of neurons are similar, are called *orientation domains*.

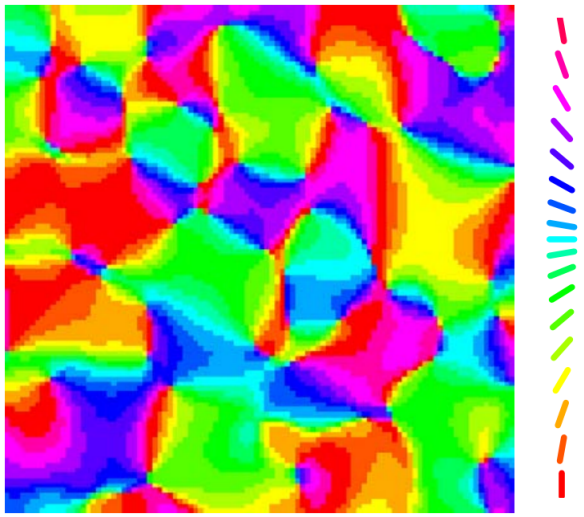

Figure 1.6: Visualization of orientation preference map. Each color corresponds to a particular angle, as shown on the right. Adapted from [Lam et al., 2005].

1.3.3 Simple and Complex Cells

Two main functional types of cells are present in the striate cortex. The first type, simple cells, detect oriented bars on an exact position in their receptive field, as explained in the previous section. In their receptive field, some areas are either excitatory or inhibitory, resulting from their response being a linear combination of ON and OFF cells on the simple cell's input. Their receptive fields behave as a Gabor filter and are therefore commonly modeled this way.

On the other hand, complex cells' output cannot be modeled simply as a linear transformation of the visual input. Their receptive field is not divided into strictly excitatory or inhibitory regions. Compared to simple cells, complex cells are invariant to the stimulus position in the receptive field. Still, they are orientation selective.

2. Deep Learning in Neural Response Prediction

This section comprises an essential background in machine learning and deep learning techniques used in neuroscience, in particular in neural response prediction. We introduce regression, deep neural networks (DNNs), convolutional neural networks (CNNs) and rotation-equivariant CNNs. To broaden the knowledge of machine learning or deep learning, we refer to two excellent books: Pattern Recognition and Machine Learning [Bishop and Nasrabadi, 2006] and Deep Learning [Goodfellow et al., 2016].

2.1 Regression

Regression is a supervised machine learning problem that is also often encountered in neuroscience, for example, in predicting a neural response given data input. Firstly, let us define what regression is.

Definition 1 (Training dataset)**.** *Let $P_{data}(Y, X)$ be a data generating distribution where Y is a random variable and $X = (X^{(1)}, \dots, X^{(d)})$ a multivariate random variable. Let $D = (\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)$ be an independently drawn sample of size n from P_{data}. We then refer to D as the training dataset. Let us also denote the empirical data distribution described by D by $\hat{P}_{data}$.*

Definition 2 (Regression)**.** *Given a training dataset D, regression is a task of finding a function $f : \mathbb{R}^d \to \mathbb{R}$ that, given $\boldsymbol{x} \in \mathbb{R}^d$, minimizes error between $\hat{y} = f(\boldsymbol{x})$ and y, where $(\boldsymbol{x}, y) \sim P_{data}$.*

Definition 3 (Loss function)**.** *Given a function $f : \mathbb{R}^d \to \mathbb{R}$, the loss function, or sometimes called cost function, is a function $L : (D \times f) \to \mathbb{R}$ estimating performance of the function f in predicting y given $\boldsymbol{x}$, where $(\boldsymbol{x}, y) \sim P_{data}(Y, X)$.*

A typical loss function for regression is Mean Squared Error (MSE). It is used not only because of its simple definition, but also because it is, in some sense, "statistically justified". More on this in Section 2.1.2.

Definition 4 (Mean Squared Error (MSE))**.** *Mean squared error loss function is defined as:*

$$L(D, f) = \frac{1}{n} \sum_{i=1}^{n} (f(\boldsymbol{x}_i) - y_i)^2, \tag{2.1}$$

where $(\boldsymbol{x}_i, y_i) \in D$ for $1 \leq i \leq n$.

Definition 5 (Hypothesis and Hypotheses space)**.** *The function $f \in F$ is commonly called a hypothesis and F a hypothesis space from which we search for the best hypothesis (for example, all linear functions, all polynomials, etc.).*

In our case, the hypothesis space is described by variables called weights (or parameters), which we will denote by w. Finding $f \in F$ is then equivalent to finding the proper weights. We will, therefore, often write $f(w)$ instead of f to

emphasize that f is parametrized by w. Moreover, we will write $L(D, w)$ instead of $L(D, f)$ since f is fully determined by w.

Regression can be solved by minimizing some appropriate loss function L. The regression task is, therefore, to find f such that $f = \text{argmin}_{g \in F} L(D, g)$.

Notice that even though we train the model using only the empirical data distribution $\hat{P}_{data}$ described by D, the task of regression is to predict $y \in \mathbb{R}$ given $\mathbf{x} \in \mathbb{R}^d$ sampled from the original data generating distribution P_{data}. To obtain a hypothesis f predicting y as accurately as possible, the size of the training dataset is, therefore, critical.

2.1.1 Irreducible error and intrinsic noise

It is often beneficial to think of random variable Y in a sample $(X, Y) \sim P_{data}$ as a sum of two random variables: $Y = Y_{reducible} + Y_{noise}$. Random variable $Y_{reducible}$ is generated by a distribution that can be fully explained by some mathematical function $g(X)$. Assuming regression with MSE loss function, the optimal hypothesis g is $g(X) = \mathbb{E}_{Y|X}[Y]$, where $(X, Y) \sim P_{data}$. On the other hand, the random variable Y_{noise} is independent of X and generated by a noise generating distribution.

This consequently leads to decomposition of the model's loss into two summands [Bishop and Nasrabadi, 2006]:

$$\mathbb{E}_{X,Y}[L] = \mathbb{E}_{X,Y}[(f(X, w) - g(X))^2)] + \mathbb{E}_{X,Y}[(g(X) - Y)^2] \tag{2.2}$$

Our model minimizes the first term, while the second term is defined by noise which the model cannot predict. For this reason, literature refers to this as an *irreducible error*, giving the lower bound of the overall loss. Note that if we did not use MSE as a loss function, $Y_{reducible}$ would be still possible to fully explain by some $g(X)$, but it would not necessarily be equal to $\mathbb{E}_{Y|X}[Y]$.

The noise can be caused by the measuring device, or more importantly, in our case, it can depend on some hidden features not provided in our dataset. The machine learning model can never predict the noise distribution. This is not caused by a poor choice of a model but by the lack of information on which the variable Y is dependent.

2.1.2 Choosing an appropriate loss function for neural response prediction

When deciding what loss function is appropriate for training of our model, the distribution of random variable Y_{noise} is of high importance as it determines the loss function. In regression, it is often assumed that Y_{noise} is generated by Gaussian distribution. Among other reasons, this decision is made mostly based on the fact that between all distributions on real numbers with mean μ and variance σ^2, the Gaussian distribution $N(\mu, \sigma^2)$ is a distribution with the highest entropy. Therefore, if we have no information about Y_{noise}, according to the principle of maximum entropy, normally distributed Y_{noise} is the best choice as it minimizes the amount of information built into the distribution. Interestingly, it can be shown that the approach of maximum likelihood estimation for $Y_{noise} \sim N(\mu, \sigma^2)$

is equivalent to minimizing mean squared error. This argument justifies the use of mean square error as a loss function for regression.

However, when predicting neural responses, normal distribution is not appropriate mainly due to the following two reasons:

1. Neuron's response is assumed to be non-negative as it correlates with its firing rate. As Gaussian distribution can generate negative values, it is an inappropriate distribution of Y_{noise}.

2. Variance of neuron's response correlates with its firing rate [Goris et al., 2014], meaning that the random variable Y_{noise} is dependent on the absolute value of $Y_{reducible}$. Using a distribution whose variance increases linearly with the mean would be beneficial.

These two observations provide additional information about the noise generating distribution, and, thus, a distribution of lower entropy can be used. A distribution consistent with the mentioned assumptions is the Poisson distribution. It is, therefore, usually used as a distribution of Y_{noise} of neural activity data. Analogically to the Gaussian distribution, using maximum likelihood estimation leads to Poisson loss, a loss function commonly used for neural population encoding prediction [Cadena et al., 2019], [Klindt et al., 2017], [Sinz et al., 2018], [Lurz et al., 2021]:

Definition 6 (Poisson loss)**.**

$$L_{Poiss}(D, f) = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i log(f(\boldsymbol{x}_i)), \tag{2.3}$$

where $(\boldsymbol{x}_i, y_i) \in D$ *for* $1 \leq i \leq n$.

2.2 Regularization and Overfitting

In machine learning, the objective is to capture all necessary patterns in the training dataset and perform well on previously unseen inputs. If the loss function contains only terms that minimize the error on training data, we might obtain a model performing perfectly on the training dataset, learning even patterns caused by noise, consequently leading to poor performance on other unseen data. This phenomenon, called *overfitting*, is a typical result of machine learning models with little or no regularization. Overfitting often occurs in the presence of a small training dataset, which is typically the case in neuroscience.

Regularization is a technique minimizing the *generalization error*, the error on previously unseen data. For this reason, among many other techniques for model regularization, new terms are usually added to the loss function to penalize functions that do not seem general enough and tend to overfit.

Usual regularization terms are the L_1 and L_2 norms of weights w, which favor simpler functions and thus prevent overfitting on the training dataset. The norms are defined as follows:

$$L_1(w) = \sum_i |w_i| \tag{2.4}$$

$$L_2(w) = \sum_i w_i^2 \tag{2.5}$$

Finally, to regularize our model, we add the regularization terms to the loss function:

$$L_{reg}(D, w) = L(D, w) + \lambda_1 L_1(w) + \lambda_2 L_2(w), \tag{2.6}$$

where λ_1 and λ_2 are parameters not tuned by the model but instead set previously by the author's choice. Parameters that have to be previously set and are not fitted by the gradient-based algorithm are called *hyperparameters*. Hyperparameters are often chosen based on the performance on the validation dataset; a separate previously unseen dataset used to perform validation of our model and to tune the hyperparameters. It is crucial that the model did not previously see this data because one of the most important information we get from the validation dataset is whether our model is overfitted or not.

2.3 Deep Neural Networks

In recent years, deep neural networks (DNNs)[1] gained respect in the computer science community due to their high performance on various tasks, outperforming many state-of-the-art solutions. Two crucial factors primarily caused DNNs to dominate previous solutions. Firstly, the higher availability of computational power enabled the world to train very deep neural networks on large datasets. Secondly, in 1989 the Universal Approximation Theorem was proven [Cybenko, 1989]. In this paper, the author proved that under certain conditions, any function $g : [0, 1]^d \to R$ can be arbitrarily closely approximated by a multilayer perceptron, an elementary type of artificial neural network. In this particular proof, the most limiting condition was a specific type of activation function along with a fixed network architecture (term defined in the next section). However, later papers broadened the classes of activation functions and DNN architectures [Leshno et al., 1993], [Heinecke et al., 2020] [Zhou, 2020], giving a solid theoretical underpinning to the whole field of deep neural networks; also called deep learning.

Deep neural networks nowadays also dominate the field of neural response prediction in the visual pathway [Klindt et al., 2017], [Lurz et al., 2021], [Ecker et al., 2018], [Butts, 2019], [Sinz et al., 2018]. Our approach builds upon the current DNN architectures. Before getting into details, let us describe an artificial neural network.

2.4 Feed-Forward Deep Neural Networks

A DNN is a machine learning model built of interconnected artificial neurons called perceptrons, units, or nodes. We limit the large class of neural networks

[1]Do not confuse with biological neural networks in the brain. Deep neural networks (also called deep artificial neural networks) were historically inspired by the human brain. One node in DNN, sometimes called a perceptron, unit, or neuron, should correspond to a simplified model of a cortical neuron [Rosenblatt, 1958]. In this work, we try to avoid the term neuron due to the confusion it might cause.

to feed-forward DNNs, networks that do not contain a cycle. In terms of already defined notions, we can think of feed-forward DNNs as a hypothesis space defined by the network's hyperparameters and weights. An artificial neuron can be described as a mathematical function that performs a weighted sum of its inputs that is passed to an activation function. This activation function is usually non-linear so the whole neuron is not just a linear mapping. Mathematically, the output of one node is computed as follows:

$$u_{out} = a\left(\sum_{i=1}^{k} w_i u_{in_i}\right), \tag{2.7}$$

where u stands for the particular neuron and w_i are the weights of the connections into the neuron u. This neuron u has k inputs.

Artificial neural networks are usually organized into layers of neurons. Since current models use many layers stacked one after another, the networks are called deep neural networks. Every layer is a function that transforms an input tensor x to an output tensor y. We refer to the connections between layers, their sizes, and other properties of the layers and the whole network as the *architecture* of the DNN model. Concrete numbers specifying the model's architecture belong to the model's hyperparameters.

A typically used deep learning layer, often called the fully connected layer, has every neuron connected to every input (usually being the neurons of the previous layer). Mathematically, the unit's j-th output is computed as follows:

$$y_j = a\left(\sum_{i=1}^{k} w_{j,i} u_{in_i}\right), \tag{2.8}$$

where $w_{j,i}$ is the weight associated with the i-th neuron from the previous layer and a is the activation function.

We obtain a basic deep neural network by stacking q fully connected layers after each other. Its output can be computed as $y = l_1(\dots l_q(x)\dots)$, where l_i is a transformation performed by the i-th layer. In these simple sequential architectures, the first layer is called the input layer, the last layer is usually refered to as the output layer, and all the layers in between are called the hidden layers (Figure 2.1).

Common activation functions used in artificial neural networks are the following:

1. **Sigmoid**: The first activation function we mention is the sigmoid function. Its application is common in binary classification as the sigmoid returns values between 0 and 1, which can be interpreted as probability of a certain class. It is, however, not used in hidden layers of deep neural networks as multiple sequentially stacked sigmoid functions cause an infamous *vanishing gradient problem*, preventing the DNNs from learning effectively [Goodfellow et al., 2016]:
$$\sigma(x) = \frac{1}{1 - e^{-x}} \tag{2.9}$$

2. **Hyperbolic tangent (tanh)**: Another popular activation function tanh, a tan function on a circle. Hyperbolic tangent can be defined using sigmoid

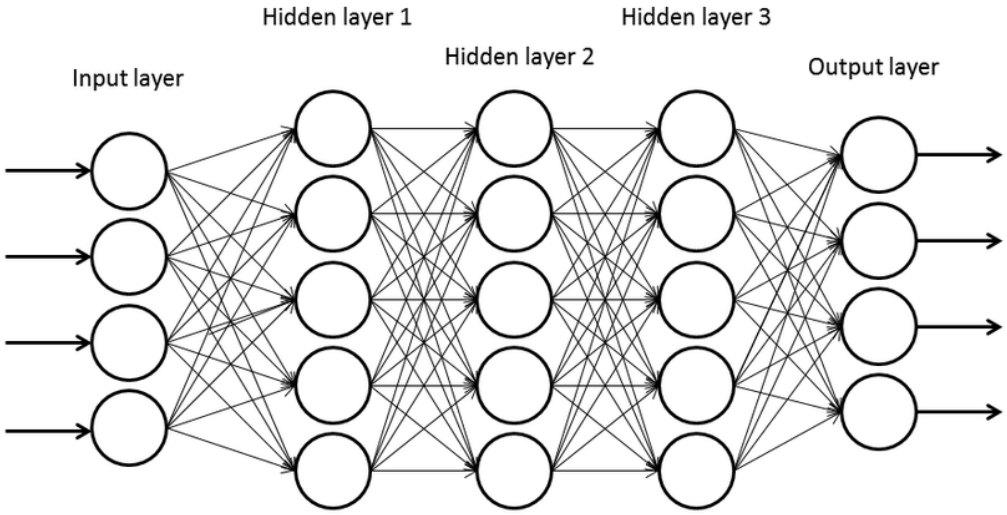

Figure 2.1: An example of sequentially stacked fully connected layers. This architecture has three hidden layers. The number of neurons in all the layers can vary. Adapted from [Miralles-Pechuán et al., 2017].

function:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.10}$$

3. **Rectified Linear Unit (ReLU)**: ReLU is beneficial for tasks where an activation function has to return positive value. It is, therefore, an appropriate choice as the last activation of the DNNs for neural encoding prediction since the final value has to be non-negative. It is, however, also used in cases where positivity is not required:

$$\text{ReLU}(x) = \max(0, x) \tag{2.11}$$

4. **Exponential Linear Unit (ELU)**: When $x \geq 0$, ELU acts as an identity, and, compared to ReLU, ELU can return negative values. The main difference between ELU and ReLU is, however, that ELU is a smooth function. Exponential Linear Unit is known to perform well and improve DNN training. [Clevert et al., 2015]:

$$\text{ELU}(x) = \left\{ \begin{array}{ll} \alpha(\exp(x) - 1), & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{array} \right\} \tag{2.12}$$

5. **AdaptiveELU**: In prediction of neural population responses, non-negativity is appropriate due to the neural responses being positive. AdaptiveELU is a non-negative activation function that has the beneficial training properties of ELU. This is achieved by shifting ELU by a predefined offset.

$$\text{AdaptiveELU}(x) = \text{ELU}(x - x_{shift}) + y_{shift} \tag{2.13}$$

6. **Softplus**: This activation function can be interpreted as a smooth ReLU.

$$\text{Softplus}(x) = \log(1 + \exp(x)) \tag{2.14}$$

2.5 Training of the DNNs

The objective of the DNN training is to find weights that minimize a chosen loss function $L(D, w)$. Due to the vast number of parameters, the volume of data, and the nonlinear nature of DNNs, direct analytical solutions are not feasible. Instead, neural networks utilize gradient-based methods.

Very simply put, gradient methods minimize an objective function iteratively. Let $L(D, w)$ be a loss function that we wish to minimize. The basic gradient-based algorithm looks similar to this pseudocode:

Algorithm 1 A gradient-based algorithm for finding the optimal weights w

```
Randomly initialize w
while L(D, w) has not converged yet do
    w ← w − α∇_w L(D, w)
end while
```

Being the optimization task in the vast majority of cases non-convex, it can depend on the starting point, i.e. how the trainable weights are initialized. It has been empirically proven that, depending on the task, certain initialization schemes can improve training and result in better models [Glorot and Bengio, 2010].

Intuitively, the model's parameters are updated in a direction that should minimize the loss. How large steps we perform depends on the α hyperparameter called the *learning rate* [2].

The crucial part is the computation of $\nabla_w L(D, w)$. As we have n data points in our training dataset D, we compute the gradient as follows:

$$\begin{aligned}
\nabla_w L(D, w) &= \nabla_w \frac{1}{n} \sum_{i=1}^{n} L((\mathbf{x}_i, y_i), w) \\
&= \nabla_w \sum_{i=1}^{n} \hat{P}_{data}(\mathbf{x}_i, y_i) L((\mathbf{x}_i, y_i), w) \\
&= \sum_{i=1}^{n} \hat{P}_{data}(\mathbf{x}_i, y_i) \nabla_w L((\mathbf{x}_i, y_i), w) \\
&= \mathbb{E}_{(\mathbf{x},y)\sim\hat{P}_{data}} [\nabla_w L((\mathbf{x}, y), w)]
\end{aligned} \tag{2.15}$$

Although the gradient can be computed directly using the whole training dataset, this is not usually done due to the large data volume. A popular stochastic gradient descent algorithm (SGD) [Kiefer and Wolfowitz, 1952] randomly selects a subsample of the training dataset called a batch; let us denote it B of m samples. Then it estimates the gradient as follows:

$$\hat{\nabla}_w L(D, w) = \frac{1}{m} \sum_{i=1}^{m} \nabla_w L((\mathbf{x}_i, y_i), w). \tag{2.16}$$

[2] Learning rate does not have to stay the same throughout the whole training. Algorithms often change α to adapt to the situation the algorithm is in. Nonetheless, gradient-based algorithms usually decrease the learning rate as the model's accuracy increases and more subtle weight updates are required.

The SGD algorithm iteratively uses all batches in the dataset to update the weights, performing one so-called epoch of the training algorithm. Epochs are repeated until the $L(D, w)$ has not converged or we run out of time or patience.

Many other algorithms build upon the stochastic gradient descent. Specifically, a popular SGD variant, which has empirically proven to perform better, is a widely used algorithm Adam [Kingma and Ba, 2014]. In each step, this algorithm adapts the learning rate and a gradient's direction. As a consequence, Adam tends to converge quicker than SGD.

2.6 Regularization for Deep Neural Networks

Besides L_1 and L_2 regularizations described in a general section about regularization, other techniques are also implemented for DNNs. The integral approach relevant to this thesis is batch normalization [Ioffe and Szegedy, 2015], which adaptively normalizes the output of a certain layer to have a mean equal to 0 and unit variance in every batch, stabilizing and accelerating the training. We should also mention dropout [Srivastava et al., 2014], which sets during training a random portion of the input to the next layer to zeros. This prevents the network from having neurons specialized for specific training examples, helping to develop a robust network.

2.7 Convolutional Neural Networks

Despite the network composed of fully connected layers being a universal approximator, when applied to an image input, it has many parameters due to the vast number of pixels. In order to decrease the dimensionality of the parameter space and take advantage of the spatial or temporal properties of the data, a new architecture has been developed; a convolutional neural network (CNN).

This sequential architecture consists of stacked convolutional layers. Each layer has a three-dimensional input of dimensions $(H \times W \times C)$, where H and W stand for the input's height and width, and C stands for the number of channels. For example, a grayscale image has just one channel, and an RGB image has channels for red, green, and blue colors; hence $C = 3$. An RGB picture with a resolution of 1080×1920 would be of size $1080 \times 1920 \times 3$. In the forward pass of the input through the network, the dimensions of the features can change in every layer.

The most prominent characteristic of the CNNs for visual input that distinguish them from other types of networks is their use of spatial properties. The prime focus is on the interactions of pixels in a local neighborhood, as their properties correlate more significantly than the properties of distant pixels. Trainable convolutional filters called kernels are thus used to capture relationships in rather small areas. They perform a linear operation called convolution.

Definition 7 (Convolution)**.** *Let I be a 3-dimensional input tensor of size $h \times w \times c$ and K be a kernel of size $m \times n \times o$. Convolution of a tensor I with a kernel K is an operation defined as:*

$$(K \star I)_{i,j,o} = \sum_{m,n,c} I_{i+m,j+n,c} K_{m,n,c,o} \tag{2.17}$$

For a better understanding, we append a visual explanation of the convolution of an RGB image with two kernels (Figure 2.2).

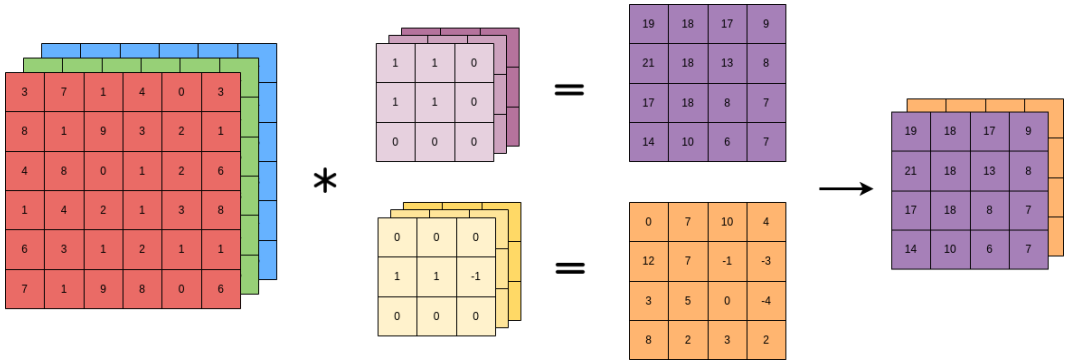

Figure 2.2: A visualization of convolution operation of $6 \times 6 \times 6$ input. Two kernels are applied on every possible pixel creating an output of shape $4 \times 4 \times 2$.

A convolutional kernel is applied in every possible position (kernel must lay inside the input tensor). Because the weights are the same during the "movement" along the input, they capture local features of the same type in every position. This reduces the number of parameters as the same properties do not have to be learned separately in different locations (the weights are shared), and, consequently, it leads to translation-equivariance, also called shift-equivariance.

Definition 8 (Equivariance)**.** *Let f and g be functions of $M \to M$. A function f is equivariant to a function g if $f(g(x)) = g(f(x))$ for every $x \in M$.*

When applied to convolution neural networks, this definition means that a CNN receiving a shifted image as input returns features transformed by the exact translation in the feature map.

In computer vision, relevant features in images often do not depend on their exact position. Because CNNs compute complex features of an input image in a position-independent way, they are the perfect choice of architecture for tasks in this field. It is thus no wonder that CNN-based architectures outperformed numerous state-of-the-art computer vision solutions of the pre-neural-network era. Their application in neuroscience is also abundant as features learned by CNNs are shown to be surprisingly similar to those in cells of the primary visual cortex [Kriegeskorte, 2015]. This fact, along with an unarguably good performance on tasks in neuroscience [Cadena et al., 2019], [Klindt et al., 2017], makes CNNs a faithful functional model of the early visual pathway and consequently an adequate mathematical model for neural response prediction in the visual system.

2.8 Rotation-Equivariant CNN (reCNN)

The foremost feature of regular CNNs is their shift equivariance property. However, a typical image transformation to which CNNs with no adjustment are vulnerable is rotation. In order to have a potent network for computer vision, it can be beneficial to have CNNs with rotation-equivariance property in addition to translation-equivariance.

For this purpose, there exist group convolutions, a class of convolutions that generalize convolutional networks to larger groups of symmetries, the rotation being among them [Cohen and Welling, 2016]. In group convolutions, the group operation is the transformation, in our case, rotation and group elements are all possible input transformations. Let us assume that we want our network to be equivariant to four orientations. Given an input image, the group elements are all its four rotations, and the group operation is a rotation by 90 degrees counter-clockwise. A graphical illustration of a group is usually used as in the following figure (Figure 2.3).

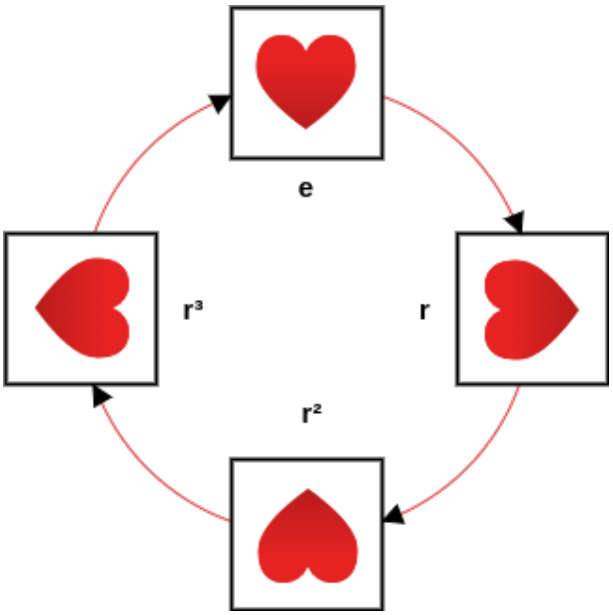

Figure 2.3: A group of 4 rotations by 90 degrees counter-clockwise. Elements are all four possible rotations depicted by a rotated heart. Arrows stand for one application of a transformation, in our case, rotation.

Firstly, a rotation-equivariant CNN creates every possible group element of the input. In our case, reCNN rotates each filter in the first layer to all allowed orientations and then applies a regular convolution with the input image. We do not rotate the input as the input is usually much larger than the filter. If we want to have 8 channels in the first layer, we obtain $4 \times 8 = 32$ new feature maps, each generated by one of eight kernels in one specific orientation.

This way, we have obtained a so-called structured feature map; features structured in groups, one for every rotation. In the subsequent layers, the computations are different. To compute one new output channel (for all rotations), a different filter is learned for every feature map from the previous layer, sticking to the assumed values, 8 filters have to be trained for every orientation. These kernels are applied to the input features, then rotated and cyclically permuted to the next group position. In this position, they are applied again, followed by another rotation and cyclic permutation. This is done for every orientation, in our case 4 times. At each element of the group, the convoluted feature maps are summed in a pointwise manner, creating an output feature for the given orientation. If we wanted to obtain more output channels, the same process would be performed with more sets of filters. Desiring 8 output channels, we would end up

with $4 \times 8 \times 8 = 256$ filters. A visual explanation is appended in the following figure (Figure 2.4).

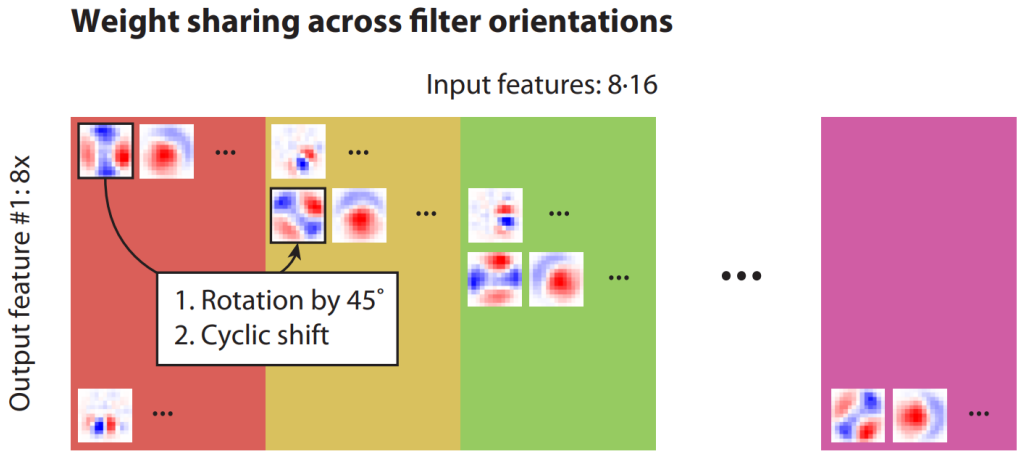

Figure 2.4: Illustration adapted from the work of Ecker et al. [Ecker et al., 2018] uses 8 rotations and 16 channels. Every colored rectangle stands for filters from which new features of one given orientation are to be created, therefore, one color stands for one orientation. The filters are rotated by 45 degrees and cyclically shifted to compute the next orientation. On the vertical dimension, the number of kernels corresponds to the number of channels on output. The horizontal dimension has size 8; one filter for every orientation of the input.

Obviously, only four orientations of a square filter are achievable without interpolation artifacts. However, as we want to have an arbitrary number of orientations, this approach is insufficient. For this reason, a steerable basis of two-dimensional functions is employed as a filter representation. In this technique, which was proposed by Weiler et al. [Weiler et al., 2018], the filter learns weights for the basis functions. As the basis is steerable, we can rotate the filter into any required orientation without interpolation artifacts.

2.9 Evaluation metrics

Although Poisson loss is suitable for the DNN training, as it depends on the scale and shift of the neural responses, it is not an ideal evaluation metric [Willeke et al., 2022]. This might complicate the comparison of the final models trained on different datasets. Moreover, being the Poisson loss not bounded, this measure is not well interpretable.

2.9.1 Averaged Pearson's correlation

Instead, we will measure Pearson's correlation coefficient per neuron between the predicted and measured responses. The averaged Pearson's correlation coefficient over all neurons will be used as the model's performance metric. Additionally, the test set contains repeated measurements of neural responses to the same visual stimulus. We will average these responses and compute the correlation of

our prediction to the average response, which will partially eliminate the neural response variability.

Definition 9 (Pearson's correlation coefficient)**.** *Given random variables X and Y, the Pearson's correlation coefficient is defined as*

$$\rho_{X,Y} = \frac{\mathbb{E}\left[(X - \mu_X)(Y - \mu_Y)\right]}{\sigma_X \sigma_Y}, \tag{2.18}$$

where μ_X and μ_Y are the means of the random variables X and Y, σ_X and σ_Y are the standard deviations of the random variables X and Y.

In comparison to Poisson loss, Pearson's correlation coefficient is a number from interval $[-1, 1]$, making this metric easier to interpret and thus more appropriate as an evaluation metric.

3. Related Work

Prediction of neural responses is a problem of many variants and parameters. We can predict neural responses in different species, different parts of the brain, different types of cells, and with different goals. The dataset concerning the V1 neural population can consist of artificial images, natural images, or videos. Most of the time, however, the responses of the primary visual cortex are predicted for static natural images. There is no benchmark to compare our model with yet, although efforts have been established recently [Willeke et al., 2022].

This chapter provides a short overview of work related to our task. Although the following publications often have different objectives than we do, they provide valuable contributions that we will take advantage of.

3.1 Prediction of Neural Responses in the Primary Visual Cortex

In the task of neural response prediction, performing an input preprocessing shared by all neurons is beneficial because neurons do not need to learn the preprocessing repeatedly, reducing the number of parameters and computational time. Antolík et al. [Antolík et al., 2016] were the first to notice and exploit this idea. Klindt et al. [Klindt et al., 2017] followed by organizing the layers of their deep neural network into two separate groups; convolutional neural network (called core in this context) and readout. Core's purpose is to generalize the computation of all neurons, disregarding their position and type.

On the other hand, the readout is designed to read preprocessed features from the core and translate them into a response of a particular neuron. Its objective is to capture only the specific properties that differ between individual neurons. In the mentioned publication, the proposed factorized readout learned for each neuron a single spatial filter of the size of the CNN's feature map's height and width. At the same time, for each neuron, the readout assigned weights to channels from the precomputed CNN features, describing the importance of each channel for a particular neuron. Regularization was imposed on all the mentioned readout parameters, causing the neuron representation to be sparse and smooth. Consequently, regularization forced the readout to predict the response only using features from a confined area of the feature map and a limited number of channels. This can be roughly interpreted as the neuron's position of the receptive field and its type.

Many papers published took into account only data from mice and not primates. Building on Klindt's work, Cadena and his colleagues decided to probe how well the spiking activity can be predicted in V1 of monkeys in response to natural images [Cadena et al., 2019].

In our work, we will use a dataset from Lurz and his colleagues [Lurz et al., 2021], where they predicted neural responses in the primary visual cortex of mice. Their goal was to investigate how well CNN core generalizes neural computation between different animals. The results were promising; indeed, the core can be used to predict neural responses in other animals, for which a lower amount of

data is available. However, the primary contribution related to our work was a novel readout architecture.

For the i-th neuron, this readout selects a feature vector $v_i \in \mathbb{R}^C$ from a certain position $p_i \in \mathbb{R}^2$ in the last layer's feature map $m \in \mathbb{R}^{C \times H \times W}$. This position is learned by fitting a bivariate Gaussian distribution $N(\mu_i, \sigma_i)$ over the feature map. During training, the position p_i is sampled from the Gaussian distribution. As the estimate of μ_i improves, σ_i shrinks. During the evaluation, μ_i is used as p_i since it is the estimate of the most relevant neuron's feature vector position. Finally, a linear combination of neuron's learned weights and the feature vector from p_i is computed to obtain the neural response, followed by a nonlinear function, in Lurz's case, ELU + 1.

In our work, an extension of this Gaussian readout will be used. A deeper description of our readout can be found in the Methodology chapter (4).

3.2 Rotation-Equivariant CNNs

As mentioned in the previous chapter, Weiler and his colleagues developed a rotation-equivariant CNN, where filters are represented as a steerable basis of two-dimensional functions, enabling the reCNNs to use an arbitrary number of orientations [Weiler et al., 2018]. The first to harness reCNN with such property on neural data were Ecker et al. [Ecker et al., 2018]. However, in comparison to Weiler's implementation, instead of circular harmonics as the steerable basis, Ecker uses two-dimensional Hermite functions of rank up to k, given the filter is of size k (overall $k(k+1)/2$ basis functions).

Ecker's study builds upon the idea presented by Klind et al. [Klindt et al., 2017]. He and his colleagues expanded the shared feature space of all recorded neurons to be independent of the orientation preference. They used reCNNs to extract every feature at a set number of different orientations. Not only does this core take into account retinotopy and, to some degree, a cell type, but it also uses orientation selectivity. In our work, we will use the same implementation of the rotation-equivariant core and slightly adjust it.

4. Methodology

In this chapter, we provide a description of the implementation, the employed external software libraries, computational resources, and an environment setup. We outline the datasets, network architecture, training process, and hyperparameter search. A repository containing the source code of our solution along with a Docker container for the training environment are publically available online[1] [2].

4.1 Used Software Libraries

We implemented the solution in a deep learning Python library PyTorch [Paszke et al., 2019] with PyTorch Lightning framework[3] to help us scale the training. NumPy [Van Der Walt et al., 2011] and Matplotlib [Hunter, 2007] libraries were used for data manipulation and visualization.

4.1.1 Nuralpredictors

The principal component we used to build our solution is the Neuralpredictors library[4] developed by Sinz Lab[5], containing the implementation of an extensive amount of neural network layers, regularizations, measure metrics, and other handy utilities designed for neural response prediction primarily in the striate cortex. Many publications related to our problem utilized and extended this library, namely Lurz et al.[Lurz et al., 2021] who implemented FullGaussian2d readout, and its isotropic three-dimensional version is also present in Neuralpredictors, on which our readout is built upon. Next significant contribution is the reimplementation of rotation-equivariant CNN [Ecker et al., 2018] from TensorFlow [Abadi et al., 2015] to PyTorch, which we exploited for the core's construction.

4.2 Datasets

For the DNN training, two distinct datasets were used. The first dataset comes from a publication from Lurz et al. [Lurz et al., 2021] who measured neural population responses to grayscale images as a visual input in a mouse's primary visual cortex. This dataset was chosen to validate the trained networks to prove that they work correctly.

The most essential dataset for this work, however, is obtained from a computational model developed by Antolík and his colleagues [Antolík et al., 2019]. Being the dataset generated in-silico, the actual locations of the neurons and their preferred orientations are available. We took advantage of this ground truth data to evaluate the precision of neural location and preferred orientation estimates.

[1] In the attachment A.1 or online at `https://github.com/mpicek/reCNN_visual_prosthesis`

[2] In the attachment A.2 or online at `https://github.com/mpicek/csng_dl_docker_image`

[3] `https://www.pytorchlightning.ai/`

[4] `https://github.com/sinzlab/neuralpredictors`

[5] `https://sinzlab.org/`

As the computational model of the striate cortex can give us virtually unlimited amounts of valuable information, the synthetic dataset is likely to be beneficial for future experiments in this area of research. Using both in-silico and experimental datasets, we can observe differences in the network's training and examine whether the dataset generated by Antolík et al. model is appropriate for stimulation protocol development.

4.2.1 Lurz et al. Dataset - The Mouse Dataset

The first dataset is a publicly available subset of data from Konstantin-Klemens Lurz and his colleagues' publication [Lurz et al., 2021], which we obtained with the authors' consent. The following text will be a paraphrased description of the data. For further details, we refer to the original publication.

The dataset is composed of pairs of visual stimuli and neural population responses. The visual stimuli are cropped grayscale images sampled from ImageNet [Deng et al., 2009], which were upsampled to the resolution of 1080×1920 and presented at a distance of 15 cm from the mouse's head. This dataset contains the images in a resolution of 64×36px as they were isotropically downsampled, corresponding to a resolution of 0.53 pixels per degree of visual angle. Although the original data have records from multiple mice, public access is restricted to records from just one subject, whose overall 5335 neurons were recorded from the striate cortex, in particular from layer L2/3. For this purpose, a wide field two-photon microscope [Sofroniew et al., 2016] was used along with a genetically encoded calcium indicator GCaMP6s in the recorded mouse. The cells were selected based on a classifier for somata [Pnevmatikakis et al., 2016]. The reported value corresponding to neural activity is an averaged cell's activity in a certain time window. Data preprocessing and stimulation paradigm were adapted from [Walker et al., 2019].

The dataset is divided into three subsets. The training set contains 4472 stimulus-response pairs and the validation set 522 pairs. To evaluate our model by measuring the correlation of averaged trials, the test set includes 990 stimulus-response pairs, of which only 99 stimuli were unique, each repeated ten times.

From now on, we will also refer to this dataset as the mouse or experimental dataset.

4.2.2 Dataset from Antolík's Model of V1 - The Synthetic Dataset

Compared to the first dataset experimentally measured on living mice, this dataset is obtained using a computational model of a cat's striate cortex developed by Antolík and his colleagues [Antolík et al., 2019]. This in-silico primary visual cortex comprises 65000 neurons modeled as exponential integrate-and-fire point neurons [Brette and Gerstner, 2005] and divided into three groups: LGN neurons, layer 4 and layer 2/3 (respectively 5000, 30000, and 30000 neurons). The cortical neurons are furthermore divided into populations of excitatory and inhibitory neurons. LGN neurons send connections only to layer 4, and cortical neurons follow specific inter-layer and intra-layer connectivity rules. As regards such rules and the value of other parameters, Antolík et al. model is constrained

population of neurons per population by a multitude of experimental findings and data measurements, making both the simulated spontaneous and visually evoked activity reliably close to the behavior of a real cat's V1. Importantly, orientation maps are also included in the model. From now on, we will refer to this dataset as the synthetic dataset.

Being this dataset produced in-silico, it is not restrained in terms of the number of trials; therefore, compared to the mouse dataset, it is much more voluminous. The pairs of visual stimuli and neural population responses contain resized and cropped grayscale images from ImageNet [Deng et al., 2009] of resolution 110×110 px and the responses of 30000 neurons from layer L2/3. Responses are obtained by summing the number of spikes within a 500 ms time window corresponding to stimulus presentation.

In this dataset, each visual stimulus spans 11 degrees of the visual field. The cortical neurons are, however, distributed in a region of the cortex corresponding to the 4 central degrees of the visual area covered by the stimuli. This was chosen to avoid boundary effects and accommodate the cortical neurons' full receptive field. The data is divided into three groups of stimulus-response pairs. The training dataset consists of 42250 samples and a validation dataset of 5000 samples. The test set comprises 5000 image-response pairs, of which 500 images are unique, each presented to the model 10 times.

This dataset, however, is not publicly available. To grant access, please, contact us.

4.2.3 Pytorch Lightning DataModules

For a convenient manipulation with the datasets, we implemented a DataModule class from the PyTorch Lightning library for both datasets separately. This facilitates straightforward data handling and its feeding into the neural networks. Our implementation is conducive for future work as it can be easily reused.

4.3 Network Architecture

From a mathematical point of view, this work aims to develop a function generalizable across neurons that, given an image of resolution $H \times W$, receptive field positions, and preferred orientations of m cortical neurons, returns a prediction of the neural population response. That is:

$$f : (\mathbb{R}^{H \times W \times 1} \times \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R}^m) \to \mathbb{R}^m, \tag{4.1}$$

where the first parameter is the grayscale input image of resolution $H \times W$, the next two parameters are the spatial coordinates x and y of each neuron's receptive field center, and the last parameter is a vector of orientation preference of every neuron. The output is the predicted neural response for each neuron in the population.

We decompose this function into two separate functions; A function $c : \mathbb{R}^{H \times W \times 1} \to \mathbb{R}^{H \times W \times O}$ represents the core. It extracts a feature map of dimensions $H \times W \times O$ from a grayscale image of dimensions $H \times W \times 1$. A function $r : (\mathbb{R}^{H \times W \times O} \times \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R}^m) \to \mathbb{R}^m$ represents the readout. Given features

from the core, spatial positions of the neuron population, and their preferred orientations, it returns an estimate of their response.

The function f is then $f(I, x, y, o) = r(c(I), x, y, o)$, where $I \in \mathbb{R}^{H \times W \times 1}$ is the input image, $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^m$ are the vectors of positions and $o \in \mathbb{R}^m$ is the vector of preferred orientations.

As a hypothesis space for finding hypothesis f, we chose deep neural networks. The function c, the core, is a convolutional neural network as the previous work proved them to be powerful. In particular, we use a rotation-equivariant convolutional neural network as a core, which extracts features for every orientation, keeping only one channel per orientation in the last layer so that we can test the prediction performance computed from just the neuron's position and its preferred orientation with as little required knowledge about the particular neurons as possible.

The readout layer, a DNN layer that models the function r, estimates the neurons' receptive field positions and their preferred orientations based on the input data. This layer's purpose is to predict the neurons' responses based on their position and preferred orientation. Each neuron's prediction is computed from a scalar value acquired by bilinear interpolation from the feature map provided by the bottleneck in the reCNN core. This interpolation is done in the spatial location defined by the neuron's position estimate, whereas the estimated orientation preference corresponds to the position in the third dimension of the feature tensor.

As the readout is constrained to retrieve only a scalar value from the core's bottleneck, its possible information processing is minimal, keeping most of the computation in the core. In this way, the response prediction for each neuron in the readout is calculated with a computation that corresponds to a specific position and a specific orientation. In this way, we estimate the lower bound of the highest achievable neural response prediction performance given the neuron's position and its preferred orientation. We propose that such a core learns an "averaged" V1 computation for every position and orientation.

The proposed DNN architecture behaving as the hypothesis f trained on the mouse dataset from Lurz et al. [Lurz et al., 2021] can be found in the Figure 4.1. The following sections present an in-depth description of the proposed core and readout.

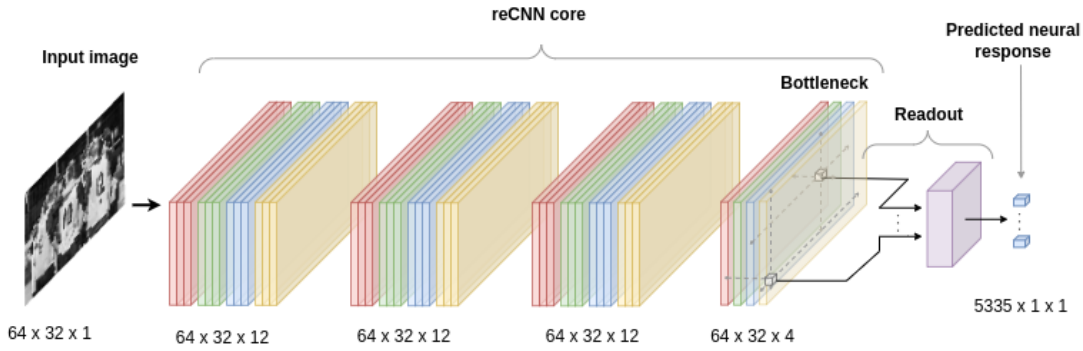

Figure 4.1: A depiction of the proposed DNN architecture for Lurz et al. dataset [Lurz et al., 2021]. An input image of resolution 64×32 px is passed into a reCNN core with 4 layers equivariant to 4 rotations. The first three layers use 3 channels, which are reduced into a single channel in the bottleneck. Each color denotes a different orientation. The readout simultaneously learns the positions and orientations of all neurons, mapping them to positive real values corresponding to the neural responses. The numbers of layers, orientations and channels are chosen for the sake of clarity, not predictive performance. The optimal hyperparameters are described in Section 5.4.

4.4 Core: A rotation-equivariant CNN with a Bottleneck

We implemented the above-described core and called it a *reCNN bottleneck* as the last layer has a single channel for every rotation. Our approach consisted of adding a reCNN layer with one channel at the end of the core. Unfortunately, the Neuralpredictors library does not provide cores with a different number of channels in every layer, so the implementation was not straightforward. Firstly, we examined the implementation provided by Sinz lab, and with this knowledge, we appended the last layer.

As a nonlinear activation function, we use ELU. After each convolutional layer, a batch normalization layer follows. Input features are padded with zeros to maintain the same feature map size during the forward pass throughout the whole network.

Inspired by the paper from Lurz et al. [Lurz et al., 2021], we also allow the networks to use depth-separable convolutional layers [Chollet, 2017] in every layer after the first one. Whether to use them was based on the performance on the validation dataset.

Speaking of the core's regularization, we utilized two forms of regularization, smoothness and group sparsity, both proposed by [Klindt et al., 2017]. Smoothness consists of the Laplacian of the convolution filters squared, that is:

$$L_{laplace} = \lambda_{laplace} \sum_{i,j,k,l} (W_{:,:,k,l} \star L)^2_{i,j}, \tag{4.2}$$

where $\lambda_{laplace}$ is a hyperparameter controlling the regularization strength, $W_{i,j,k,l}$ is the convolution kernel, i, j are the spatial dimensions, and k and l

are the input and output channels. Matrix L is the Laplacian filter and is defined as follows:

$$L = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 1 & -6 & 1 \\ 0.5 & 1 & 0.5 \end{bmatrix} \tag{4.3}$$

Group sparsity regularization is defined as:

$$L_{group} = \lambda_{group} \sum_{i,j} \sqrt{\sum_{k,l} W_{i,j,k,l}^2}, \tag{4.4}$$

where λ_{group} is the hyperparameter controlling the regularization's importance, and $W_{i,j,k,l}$ is the convolution kernel described in the previous regularization term. Group sparsity forces the kernels to use a smaller number of features from the previous layer.

4.5 Readout: A Three-Dimensional Gaussian Readout with a Cyclic Last Dimension

Our second contribution is the readout. Similar to the readout proposed by Lurz et al. [Lurz et al., 2021], our readout layer estimates each neuron's receptive field position in the core's feature map, from which it subsequently fetches value and translates it into a neural response value. A three-dimensional extension of this readout is also available in the Neuralpredictors library. In addition to learning a neuron's spatial position, it decides on one channel from which the value is yielded.

We extended the third dimension to be periodic so that the strict interval mapping onto the orientation features does not impede the network's training. Moreover, this cyclicity allows for a bilinear interpolation between the first and the last orientation features.

Regarding the implementation, whenever the readout reads a value from the third dimension out of the specified range $[-1, 1]$, the read position in this dimension is periodically shifted back into the interval $[-1, 1]$ and then mapped onto the feature map's third dimension. The read value is obtained by bilinear interpolating the corresponding feature map's units.

To impose periodicity on interval $[-1, 1]$, the position in the orientation dimension was adjusted in the following way:

$$o_{periodic} = ((o_{position} + 1) \bmod 2) - 1 \tag{4.5}$$

This shifted the interval to $[0, 2]$, the modulo operation moved the position into the interval, and then we shifted the interval back to $[-1, 1]$. Concerning the first two dimensions, the positions to read the features from are clamped into the interval $[-1, 1]$. This is because retinotopy does not behave periodically; therefore, cyclicity is inappropriate in learning a neuron's position of its receptive field.

Although the above modification of the three-dimensional Gaussian readout samples values from features mapped to the interval $[-1, 1]$, it is not periodic yet.

The orientation of value mapped to 1 does not correspond to the orientation of value in the feature map mapped to value -1. The first value corresponds to an orientation of 0 degrees, and the last corresponds to $360\,(O-1)/O$ degrees, where O stands for the number of rotations. If we wanted to acquire value for orientation r such that $360\,(O-1)/O < r < 360 = 0$ degrees, the readout would not be able to interpolate between $360\,(O-1)/O$ and $360 = 0$ degrees. For example, if we had four possible orientations, the third dimension of the last reCNN core's layer would have a size of 4, and the following degrees would be mapped to the values from $[-1, 1]$:

Value	Degree
-1	0
$-0.\overline{3}$	90
$0.\overline{3}$	180
1	270

Table 4.1: Table of values and their corresponding degrees.

We copied all values from the feature map corresponding to an orientation of 0 degrees after the last layer to fix this issue. Consequently, the readout can properly fetch values from the core's features employing bilinear interpolation even for values r such that $360\,(O-1)/O < r < 360 = 0$ degrees. Sticking to numbers from above, we obtain this mapping of the interval $[-1, 1]$ to the orientation degrees:

Value	Degree
-1	0
-0.5	90
0	180
0.5	270
1	360

Table 4.2: Fixed mapping of values to the corresponding degrees.

The interpolated value acquired by the readout is multiplied by a learned factor and shifted by a bias, which is initialized to a mean response of a particular neuron. Consequently, the value is passed into a Softplus activation function described in the chapter about feed-forward DNNs (Equation 2.14), yielding the final neuron's response prediction. Since the readout is restricted to using just one position from the bottleneck, no regularization is imposed on the readout layer.

4.6 Training Pipeline

The process of finding the best model does not consist solely of a gradient-based algorithm for finding optimal model weights. In this section, we describe our training pipeline, including packaging the model, setting it up on remote servers, hyperparameter search, model training, and eventually evaluating and reusing the final trained model.

4.6.1 Environment and Computational Resources

To train the network, proper hardware with sufficient computational power is required. It is usually beneficial to containerize the solution into a software package that can be easily scaled and run on multiple machines.

We opted for Docker, a tool providing such containerization [Merkel et al., 2014]. We built a Docker image based on available Nvidia NGC containers[6] with installed PyTorch library and support for CUDA GPUs [NVIDIA et al., 2020] with other additional libraries, for instance, Neuralpredictors.

Due to a large number of parameters, the DNN's training is computationally demanding. In our work, we harnessed resources provided by MetaCentrum[7]. Because remote virtual machines obtained from a scheduling system on MetaCentrum use Singularity [Kurtzer et al., 2017] instead of Docker, our Docker image had to be converted into an image compatible with Singularity. Fortunately, Singularity is prepared for this situation and made this conversion straightforward. Additionally, we developed multiple scripts simplifying the model's deployment on the MetaCentrum infrastructure.

4.6.2 Hyperparameter Search

In deep learning, finding the optimal hyperparameters is a crucial problem. The model's author can either strictly set the hyperparameters according to previous experience, or some other algorithm is employed to search in a hyperparameter space, trying different solutions and using the best performing one on the validation set. During the hyperparameter search, it is favorable to track the experiments and model versions, visualize the results and later reproduce the models.

For these purposes, we utilized a machine learning tool Weights and Biases [Biewald, 2020], providing all the features mentioned above, which came in handy during the development. As regards the hyperparameter search, it can be done in multiple distinct ways. The most commonly used search techniques, which are also supported by Weights and Biases, are the following:

1. Choose hyperparameter values randomly (random search).
2. Given a finite hyperparameter space, try all possible hyperparameter values (grid search).
3. Search for the optimal hyperparameters in a Bayesian fashion [Snoek et al., 2012].

We preferred the first and the last technique. The second option is not feasible in our case due to the infinite number of parameters to be set. Even discretizing the space of values that hyperparameters can take, due to the high dimensionality of it, covering it extensively is computationally unfeasible. At the beginning of our hyperparameter search, we employed a random search to get a gist of how the hyperparameter space behaves and weed out hyperparameters that do not yield a capable model.

[6] `https://catalog.ngc.nvidia.com/`

[7] `https://www.metacentrum.cz/en/index.html`

Based on these empirical results, we narrowed the possible ranges of hyperparameter values to choose from, obtaining a smaller hyperparameter subspace. Consequently, we harnessed the Bayesian search to deliberately select the best hyperparameters. This technique prefers with some probability to select hyperparameters proven to perform well on the validation dataset based on the previous trials.

Weights and Biases provide a helpful feature for hyperparameter search called Sweeps. After a Sweep is initialized, virtual machines can connect to the Weights and Biases platform to acquire the next set of hyperparameters on which a network should be trained. Metrics, losses, and other valuable information are logged into the platform during training and can be consequently visualized in a web application provided by Weights and Biases.

4.6.3 Control Model

Only reporting the achieved values of the evaluation measures defined in Section 2.9 does not give the reader any gist of how using the bottleneck decreases the predictive performance. To give more context to our achieved performance, we developed and trained a model proposed in the paper from Lurz et al. [Lurz et al., 2021], mainly because they aim to create a core that generalizes well between different animals, making a perfect architecture for this comparison. In contrast with our model, Lurz and colleagues want to achieve the highest predictive performance possible with no restrictions. In contrast, our network has its computation strictly constrained by the bottleneck layer at the end of the reCNN core. It is, therefore, expected that our network will have much lower predictive performance. The question is, however, how much the performance drops.

For this reason, we compute the correlation fraction with respect to the control model. Specifically, this metric is obtained as the fraction of the correlation achieved by the bottleneck model over the correlation of the control model predictions. As regards the correlation used, we correlate each model's predictions with the averaged response of the repeated trials. This result gives more information about the decrease in accuracy due to the bottleneck's presence and a readout that uses just a scalar value for a neuron's response prediction.

This evaluation technique has, however, also an important drawback that needs to be taken into account. The control model's quality influences the reported fraction of achieved performance. The worse the control model is, the better our bottleneck model seems to be.

5. Experiments and Results

This chapter aims to outline the performed experiments on both datasets along with achieved results. In the discussion section, we interpret the results and explain what might be altered or improved in future experiments. Finally, we propose future experiments that may follow the present work.

5.1 Finding the Optimal Model

As described in the last chapter, we utilized Weights and Biases platform for a hyperparameter search. We hypothesized that the model's performance is highly dependent on the number of orientations to which the network should be equivariant. As the readout is constrained to predict a neuron's response from just one scalar value, it needs to fetch the number from a relevant spatial position and a relevant orientation. The more orientations are available to choose from, the more specialized the features can potentially be, and thus the performance should increase. To investigate this hypothesis, each hyperparameter search has the number of orientations fixed, dividing the network architectures into groups determined by the training dataset and number of orientations to which the network should be equivariant. From now on, we will refer to these groups of networks as *categories*. Moreover, with a higher number of orientations, the number of parameters in the core increases linearly, leading to a significant increase in the number of trainable parameters to be fit. This is another reason to search for hyperparameters separately for a distinct number of orientations.

The models were fitted by the Adam optimizer [Kingma and Ba, 2014] with an early stopping after 5 or 10 consecutive steps during which the correlation on the validation dataset did not improve (10 in the case of the Lurz et al. dataset [Lurz et al., 2021], 5 in case of the synthetic dataset). The batch size was set to 10. The model's performance was evaluated by Pearson's correlation on the validation dataset and was consequently used for comparing different choices of hyperparameters. The best-performing model was evaluated on the test set by Pearson's correlation of the predicted responses with ten averaged responses to the same stimulus.

5.2 Searched Hyperparameters

An important parameter was the learning rate, which controls how strongly the gradient influences the weights' update in each step. The next three hyperparameters were kernel sizes in the first layer, the last layer, and all other layers. Although the number of layers and channels in each convolutional layer are dependent variables, they were tuned independently of each other as the Weights and Biases platform does not yet support such a feature. The core's regularization strength factors in the first layer and the other layers were the next values to be found by the hyperparameter search. The proper way to initialize the network was learned by tuning the ranges of initial neurons' positions and by the parameter sigma describing the standard deviation of the Gaussian curve,

from which the neurons' positions were sampled. Another question was whether to use depth-separable convolution or not, which was also decided based on the performance on the validation dataset.

5.3 Control Model

In the control model adapted from Lurz et al. [Lurz et al., 2021], all layers were regularized according to the original paper. Hyperparameter searches were done on each dataset separately to find the optimal network settings. In the following sections, this control model will indicate how much predictive precision was lost due to the bottleneck.

Since our hyperparameter search is not as extensive as in the work of Lurz et al. [Lurz et al., 2021], the results are not as high. Moreover, our dataset is much more limited than the one used in the original paper. For this reason, achievable performance is likely higher. However, all trained networks had approximately similarly extensive search. If we devoted more time to parameter search of each network, the performances of all the networks in this work would likely grow proportionally, keeping the fraction of correlation intact.

5.4 Best Models Trained on Lurz et al. Dataset

We performed the hyperparameter searches for three fixed numbers of orientations; 8, 16, and 24. As hypothesized, the higher the number of rotations, the better the predictive performance of the best model. Along with the mentioned hyperparameters, whether to normalize the input images to have 0 mean and unit variance was decided based on the hyperparameter search. We present the results of the best network in the following table:

Model category	**val corr**	**test corr**	**corr fraction**	**Number of core parameters**
reCNN_bottleneck8	0.1594	0.2877	0.6106	2690552
reCNN_bottleneck16	0.1640	0.2996	0.6359	41862
reCNN_bottleneck24	**0.1750**	**0.3133**	**0.6651**	**191062**
control model	0.2645	0.4711	1.0	9325

Table 5.1: A table presenting the best-performing models for each category trained on the Lurz et al. dataset [Lurz et al., 2021]. Each column represents one metric. The metrics are in the following order, beginning from the second column: the correlation on the validation dataset, the correlation on the test dataset with the averaged response of repeated trials, the correlation fraction with respect to the control model. The last column is self-explanatory.

For completeness, the following table (Table 5.2) provides the selected values

of all searched hyperparameters of the best-performing model for Lurz et al. dataset [Lurz et al., 2021].

Hyperparameter	**Optimal value**
Whether to normalize images	False
Kernel size in bottleneck	11
Kernel size in hidden layers	15
Kernel size in input layer	13
Core regularization strength in the hidden layers	0.020269349701148538
Core regularization strength in the input layer	0.18353679655652372
Number of hidden channels	4
Number of core layers (without bottleneck)	5
Whether to use depth-separable convolutions	True
Whether sigma is fixed at the beginning of the training	True
μ range at initialization	$[-0.4, 0.4]$
σ range at initialization	$[-0.4, 0.4]$
Learning rate	0.001

Table 5.2: A table with the searched hyperparameters and the best values found for the best model trained on the Lurz et al. dataset [Lurz et al., 2021].

5.5 Best Models Trained on the Synthetic Dataset Generated from In-Silico Model of Cat V1 (Antolík et al. model)

The dataset from Antolík's model was preprocessed by normalizing the input images to have 0 mean and unit variance. Due to a large number of training examples, the hyperparameter sweeps were not that extensive, and, with more computational resources, a higher predictive performance might have been achieved.

Compared to Lurz et al. dataset [Lurz et al., 2021], sweeps were performed only for 8 and 16 orientations as we did not have computational resources to run the network with a larger number of orientations. The following table presents the achieved results:

Model category	**val corr**	**test corr**	**corr fraction**	**Number of core parameters**
reCNN_bottleneck16	0.2245	0.4795	0.7924	191122
reCNN_bottleneck8	**0.2363**	**0.5038**	**0.8327**	**109650**
control model	0.2820	0.6051	100	2690552

Table 5.3: A table presenting the best-performing models for each category trained on the dataset generated by a computational model from Antolík et al. [Antolík et al., 2019]. Each column represents one metric. The metrics are in the following order, beginning from the second column: the correlation on the validation dataset, the correlation on the test dataset with the averaged response of repeated trials, the correlation fraction with respect to the control model. The last column is self-explanatory.

Again, for completeness, we provide a table with values for all hyperparameters of the best-performing model for the synthetic dataset (Table 5.4).

Hyperparameter	**Optimal value**
Kernel size in bottleneck	15
Kernel size in hidden layers	11
Kernel size in input layer	11
Core regularization strength in the hidden layers	0.0019221631986646537
Core regularization strength in the input layer	0.016823707389172524
Number of hidden channels	8
Number of core layers (without bottleneck)	4
Whether to use depth-separable convolutions	True
Whether sigma is fixed at the beginning of the training	True
μ range at initialization	$[-0.7, 0.7]$
σ range at initialization	$[-0.7, 0.7]$
Learning rate	0.001

Table 5.4: A table with the searched hyperparameters and the best values found for the best model trained on the synthetic dataset generated by Antolík et al. model [Antolík et al., 2019].

5.6 Reconstructing the Orientation Maps

As the readout estimates the neurons' positions and their preferred orientations to subsequently read a corresponding feature from the core, given a particular dataset, we visualized the estimated positions of neurons with their predicted orientation preference. As regards the visualization of the estimates of the in-silico cat's primary visual cortex, we obtained a reconstruction of its orientation map (Figure 5.2). Moreover, as in the case of the synthetic dataset from Antolík et al. model [Antolík et al., 2019] the positions and preferred orientations are available, we visualized the original orientation map (Figure 5.3) and computed errors in the neural location and orientation preference estimates (figures 5.4, 5.5).

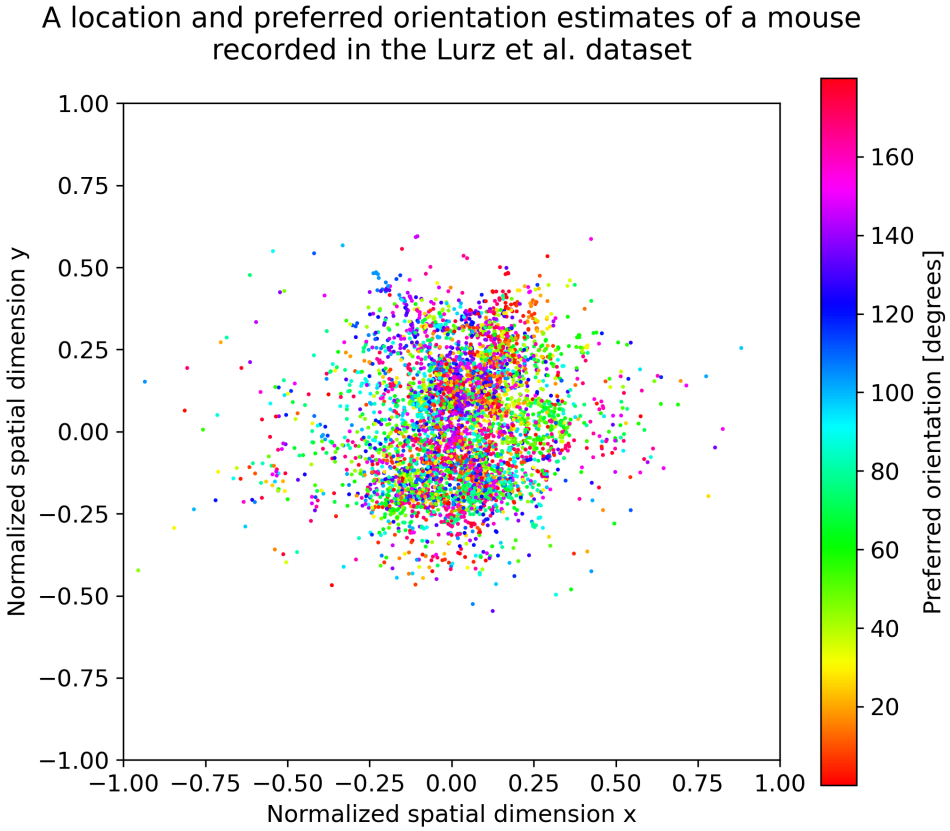

Figure 5.1: A visualization of estimated normalized neural locations along with their orientation preference of the recorded mouse from the dataset from Lurz et al. [Lurz et al., 2021]. Due to the absence of orientation maps in rodents, the plotted neurons do not exhibit any visible structure.

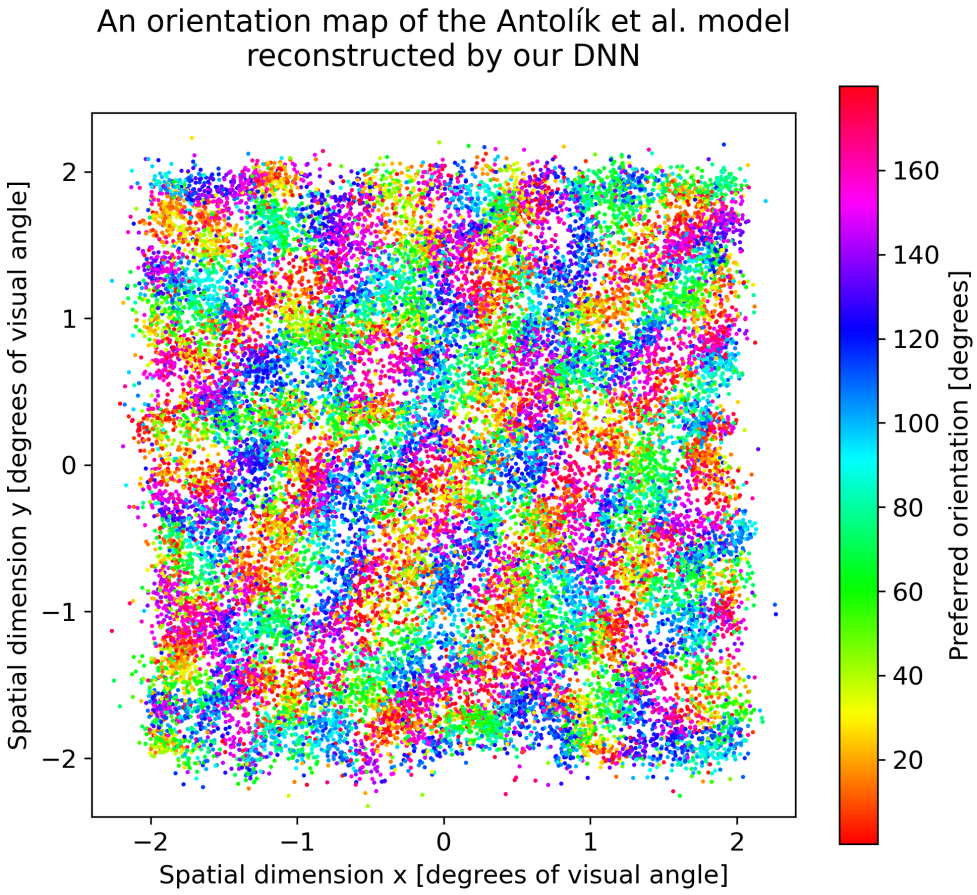

Figure 5.2: The orientation map of the Antolík et al. model [Antolík et al., 2019] reconstructed by our DNN.

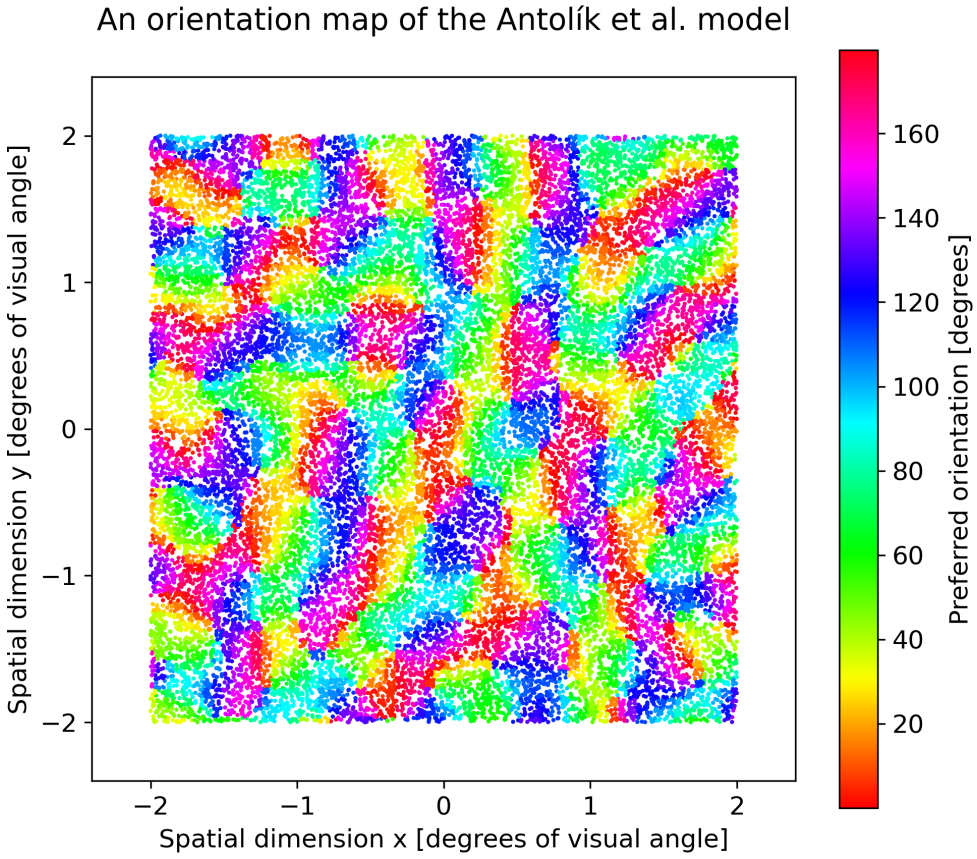

Figure 5.3: The original orientation map of the Antolík et al. model [Antolík et al., 2019].

The graphical demonstrations contain remarkable similarities in comparing the reconstructed orientation map of the in-silico cat V1 and its available ground truth orientation map (Figure 5.3). However, to adequately prove that the orientation map reconstruction is accurate, having all necessary data available, we computed the average error in both location and preferred orientation estimates.

To examine the accuracy of the readout estimates in more detail, we plotted the distribution of errors in the neural location prediction (Figure 5.4). A small portion of neurons (0.16%, in particular, 47 neurons out of an overall 30000) having an estimation error higher than 0.55 degrees of visual angle were pronounced to be outliers and disregarded in the figure not to pollute the visualization of the error distribution. Surprisingly, the average error is small; only 0.1 degrees of visual angle.

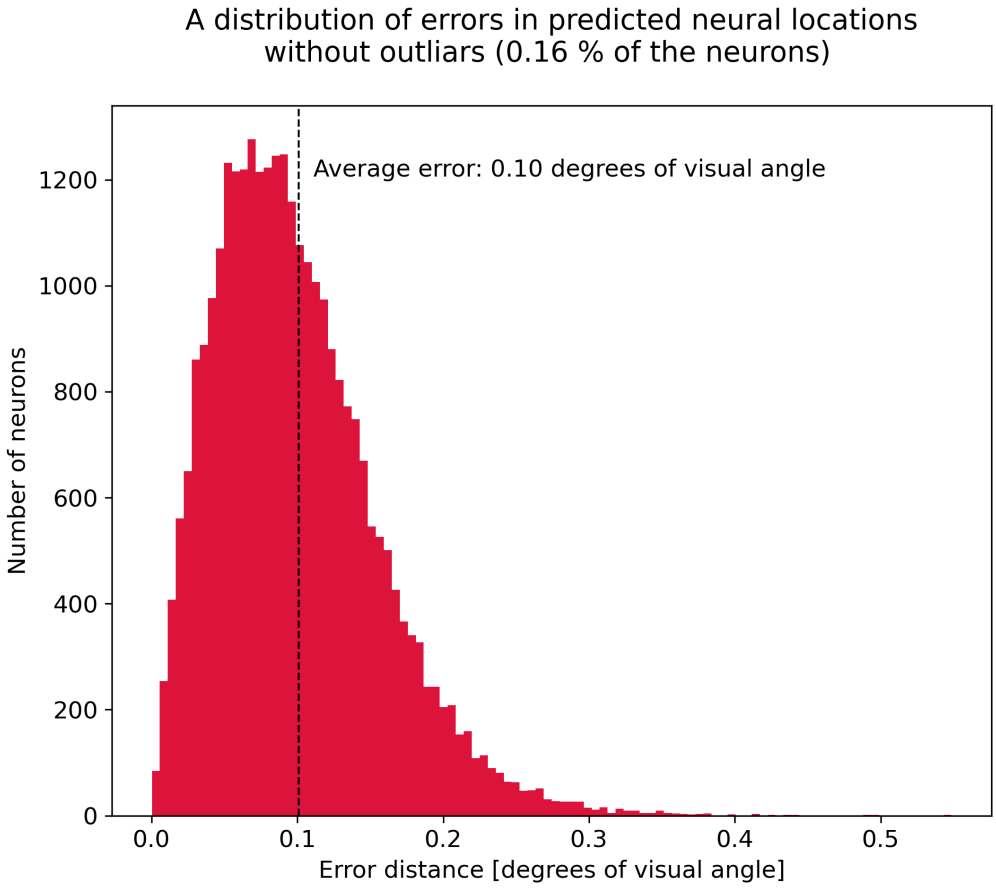

Figure 5.4: A distribution of errors in predicted neural locations that disregards 47 outlying neurons (0.16%) to not pollute the visualization.

Furthermore, we investigated the orientation preference estimates. In comparison to the predicted location, the average error is comparably higher, yielding the value of 15.93 degrees. In Figure 5.5 we present the graphical depiction of the distribution of errors in orientation preference estimates.

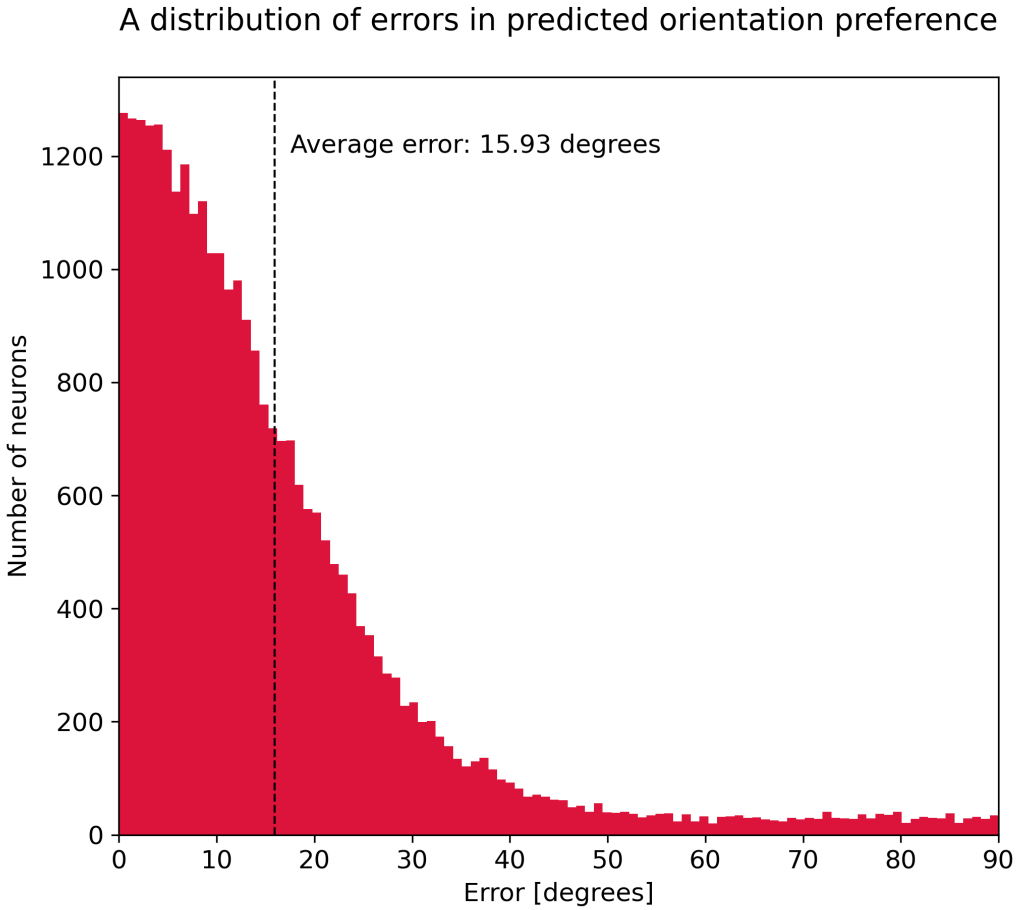

Figure 5.5: A distribution of errors in orientation preference estimates. The average error is higher compared to the error in location estimates. For a very small portion of neurons, the orientation is, however, not learned at all.

6. Discussion

6.1 Initialization Matters

During multiple hyperparameter searches performed, an interesting phenomenon emerged. If the initial neuron positions and orientations are far from the truth and if the initial standard deviation of the Gaussian distribution is too large, the network does not converge. The network usually converges and performs well if the initial neural positions and orientations are in the range of $[-0.7, 0.7]$ or smaller with a standard deviation below approximately 0.5. Moreover, networks with the same architectures and hyperparameters trained with a different seed (and, therefore, different initial positions and orientations of the neurons) were significantly inconsistent in their performance (Figure 6.1). The motives behind such high-performance variability are likely to be connected to the initialization of the readout parameters. Further research aimed at better identifying such motives could be performed in the future and possibly produce more consistently optimal performances.

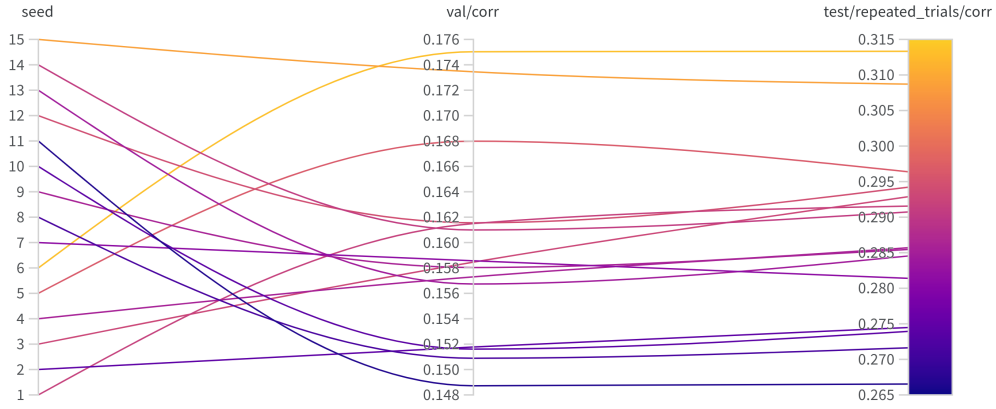

Figure 6.1: This figure represents the results of one of the best-performing network architectures trained on the mouse dataset with the number of orientations set to 24. The hyperparameters are shared between all seeds. Each line represents a result of a model with a particular seed. Clearly, initialization significantly influences the network's performance.

6.2 Different Datasets Yield Different Results

The models exhibited interesting differences when comparing the results obtained from the two different datasets.

The correlation fraction with respect to the control model achieved on the synthetic dataset was higher than the one obtained with the experimental dataset. This result is, however, expected. Although Antolík's model [Antolík et al., 2019] is a very accurate model of a cat V1, being a model, it still makes simplifying

abstractions regarding specific details of the V1 visual information encoding. One example of such simplification is, for instance, the absence of direction selectivity. Another example, for instance, is the absence of other brain areas that are known to influence V1 activity in biological systems. For this reason, the in-silico model is likely to perform a simpler information encoding and present a lower degree of intrinsic noise than the experimental dataset from Lurz et al. [Lurz et al., 2021]. As a consequence, the performance drop with respect to the control model was lower in the dataset generated by the computational model.

6.3 Lack of Computational Resources

We would likely be able to achieve even higher prediction performance with more computational power. The computation time on MetaCentrum machines with GPU is limited to a maximum length of 24 hours. This was a limiting factor for networks trained on the synthetic dataset, leading to under-fitted models, sometimes misleading the Bayesian search into hyperparameter subspaces with non-optimal hyperparameters. Moreover, if more sweeps were performed, gradually narrowing the hyperparameters space, we would probably be able to find hyperparameters that would conceive even better models.

Despite the lack of computational resources, we designed and trained deep neural networks that achieve a very high performance relative to the control model based on the state-of-the-art models for predicting neural responses. Specifically, the best model trained on the synthetic dataset generated by Antolík et al. model [Antolík et al., 2019] achieved a 0.8327 correlation fraction with respect to the control model. In contrast, the best model trained on the experimental dataset from Lurz et al. [Lurz et al., 2021] achieved 0.6651 of the same measure. While the control architecture predicts a neuron's response based on a whole vector of features, our model is restrained to using just a scalar value from the reCNN core. This bottleneck reduces the predictive performance. Based on the results, the bottleneck influenced models trained on the mouse dataset more than the models trained on the in-silico dataset, as discussed in the previous section.

6.4 Reconstruction of orientation map of the Antolík et al. model

Due to the surprisingly accurate estimates of the neural locations and their preferred orientations, we reconstructed the orientation map of the Antolík et al. model [Antolík et al., 2019]. Moreover, we examined important statistics of this reconstruction. Namely, the average error of the location estimate was 0.1 degrees of visual angle, while the average error of the preferred orientation estimate was 15.93 degrees [1].

Interestingly, in the distribution of errors of position estimates, a small portion of neurons (0.16%) exhibited a significantly larger error than the remaining neurons' estimates. In future work, it would be interesting to investigate the cause of this problem.

[1] Note that the units are different!

Although the reconstructed orientation map was very accurate, being it not our primary objective, we did not optimize our network to predict the neural locations and preferred orientations. If we had done so, the resulting orientation map might have been of even higher quality. Therefore, we propose to use our approach and slightly alter the network's behavior to focus not on the predictive neural response performance but on the accuracy of the location and preferred orientation estimates. This could be achieved by constraining the kernel sizes in the reCNN core in such a way that we obtain features in the bottleneck layer with a receptive area spanning the same degree of visual angle as the receptive field of the actual neurons in V1 of the species on which the data was obtained. This way, we might be able to model the primary visual cortex more plausibly with regard to the characteristics of certain species. Consequently, the architecture might estimate the neurons' positions and orientation preferences with higher precision.

As regards the higher error in prediction of the preferred orientation, it might have been caused by a lower orientation selectivity of a certain neuron. If the particular neuron is not very orientation selective, it is unclear what the exact orientation preference is; therefore, the predicted preference is inaccurate.

6.5 Hyperparameters of the Best Models

The best networks tend to prefer relatively large kernels (the best models on both datasets had all kernels of sizes larger than 10). It would be interesting to investigate how large the area of interest of the final bottleneck layer is and compare it to the size of the receptive field of real neurons in the primary visual cortex of a mouse or a cat.

Both models chose the learning rate of 0.001 and a small number of channels (4 and 8, respectively). The fewer channels were likely to be caused by the number of orientations that linearly increase the number of core parameters, leading to a higher learning capacity sufficient to learn all necessary patterns in the training dataset.

Conclusion

Due to the lack of high-resolution intracortical brain stimulation techniques, current research in V1 cortical prosthetics for sight restoration takes into account only retinotopy. However, stimulation devices are evolving quickly, and in the near future, their stimulation resolution might increase to such a degree that targeting separate orientation columns might be possible. As a result, the orientation preference of individual orientation columns could be exploited in designing a prosthetic stimulation protocol.

Under the underlying assumption that the representation of the visual stimulus in the cortical population activity determines the perception, the problem of brain stimulation is tightly connected to the problem of prediction of neural responses given an input visual stimulus. To stimulate the primary visual cortex, we first need to predict the population activity that would elicit the desired percept. Consequently, the stimulation device will try to achieve this neural activity in the stimulated tissue.

In this work, we focused on the achievable predictive performance of a model that predicts a neural population response given only the orientation position preference of the targeted cortical neurons. For this purpose, we developed a specialized deep neural network based on a rotation-equivariant CNN, extracting features in multiple different orientations and, thus, giving birth to a feature map not only for specific positions, as is common in regular CNNs, but also specific orientations.

Having a feature map produced by a computation resembling the visual stimulus processing of the early visual pathway up to the primary visual cortex, we can use these features for a stimulation protocol targeting the V1 cortical tissue. Given a visual stimulus, each scalar value from the feature map represents the neural activity that the V1 would naturally exhibit at a specific location of the striate cortex, having a particular orientation preference. Therefore, to stimulate a given V1 cortical column with a specific preferred orientation, we can use the feature map's corresponding scalar feature to translate it into the concrete stimulation protocol in the hope of eliciting the same neural activity the DNN model predicted.

The readout, the last layer of the proposed deep neural network, estimates the neural locations and their preferred orientations to subsequently use these estimates to read an appropriate scalar feature from the core's last layer and translate it into the neural response. New opportunities emerge by obtaining estimates of neural locations and orientation preferences. We took advantage of these predictions and used them to reconstruct the orientation map of the in-silico cat LGN and V1 model from Antolík and his colleagues [Antolík et al., 2019]. The reconstruction was surprisingly accurate and represents a novel approach to predicting orientation preference and position of neurons from a dataset of neural responses to natural images (and not ad-hoc selected stimuli for the purpose).

Moreover, we tested the network implementation on an experimental dataset from Lurz et al. [Lurz et al., 2021] to confirm that the DNN architecture also works on the experimentally obtained data. This verification was, however, also essential to demonstrate that the dataset generated in-silico does not behave too

differently compared to the experimental data, proving to be a reliable dataset for future experiments in this area of research.

Future Work

With our contribution, multiple directions for extending the current state of the research have opened up. In this last section, we provide the next problems and experiments worth exploring.

Reconstruction of Orientation Maps In-Vivo

Having reconstructed an orientation map from an in-silico primary visual cortex, we should move to experimental data from species that possess orientation maps to assess the quality of the readout estimation performance on experimental data. As already discussed in Section 6.2, the dataset generated by a computational model developed by Antolík and his colleagues [Antolík et al., 2019] might have abstract tiny nuances of the real V1 features, resulting in a dataset with a lower amount of intrinsic noise.

The Core's Generalizing Properties

A common problem in neuroscience is the lack of data. As there is little or no data about the patient's targeted cortical tissue, the least individual configuration of the prosthesis will be required to implant a well-performing intracortical stimulation device. For this reason, the provided core features should capture the most general computations of the primary visual cortex.

As Lurz et al. showed [Lurz et al., 2021], a CNN-based core effectively generalizes the computation between different animals. The remaining question is whether architectures composed of rotation-equivariant convolutional neural network and a bottleneck also possess this property.

Since the lack of data will be encountered in the clinical application of cortical prostheses, addressing the generalization property of stimulation protocol will benefit the future intracortical prosthetic industry.

Stimulation Protocol Development

The next stage we head towards in our efforts is designing the desired stimulation protocol taking into account the property of orientation selectivity of the neural tissue, provided we have at our disposal a high-resolution stimulation technique with precision high enough to target separate cortical columns along with a knowledge of the orientation map.

The proposed deep neural network with a core composed of a rotation-equivariant CNN and a bottleneck can be used for those purposes. Complex and reliable computational models of the early visual pathway are available [Antolik et al., 2021] and have already been leveraged as a simulated environment for testing the stimulation protocols. We aim to do exactly the same, specifically testing the orientation targeting stimulation protocol against a more straightforward stimulation protocol considering only retinotopy. Specifically, we'll validate the approach by

comparing the neural activity evoked by the two different protocols and the one evoked under normal vision conditions. Being these experiments done in-silico, they can be very flexible and enriching in terms of providing insight into the behavior of the cortical circuit upon which the stimulation was performed.

After this period of development in a virtual environment, research might move to living subjects, eventually even to human patients with acquired blindness. We believe this approach might lead to some degree of sight restoration for the blind.

Video

A stimulation protocol based solely on static images will not be sufficient to develop a cortical implant inducing high-quality visual percepts. A practical prosthetic device will need to have natural videos as input. An exploration in this direction is needed as not many studies have addressed this issue yet [Sinz et al., 2018].

Although in-vivo experiments with video are difficult and expensive, the first model architectures could be proposed based on the data generated in silico from models of the V1, exploiting the same approach of the dataset here used. Subsequently, the neural predicting models trained on responses to videos could be verified on experimental data and compared to models trained on static images.

Account for Other Inputs to V1

The work of Sinz and his colleagues proved that accounting for external information correlated to the brain state improves the model's predictive performance [Sinz et al., 2018], [Bashiri et al., 2021]. Since wearable devices measuring such data are readily available (e.g., smartwatches), they could be leveraged to improve the quality of the subject's percepts. This approach is biologically justified because the primary visual cortex has connections from other brain areas, influencing the neural population encoding (Figure 1.2).

Currently, however, there is not enough experimental data to work with, making this area of research difficult.

Bibliography

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, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Peter Ackland, Serge Resnikoff, and Rupert Bourne. World blindness and visual impairment: despite many successes, the problem is growing. *Community eye health*, 30(100):71, 2017.

Ján Antolík, Sonja B Hofer, James A Bednar, and Thomas D Mrsic-Flogel. Model constrained by visual hierarchy improves prediction of neural responses to natural scenes. *PLoS computational biology*, 12(6):e1004927, 2016.

Ján Antolík, Cyril Monier, Yves Frégnac, and Andrew P Davison. A comprehensive data-driven model of cat primary visual cortex. *BioRxiv*, page 416156, 2019.

Jan Antolik, Quentin Sabatier, Charlie Galle, Yves Frégnac, and Ryad Benosman. Assessment of optogenetically-driven strategies for prosthetic restoration of cortical vision in large-scale neural simulation of v1. *Scientific reports*, 11 (1):1–18, 2021.

Mohammad Bashiri, Edgar Walker, Konstantin-Klemens Lurz, Akshay Jagadish, Taliah Muhammad, Zhiwei Ding, Zhuokun Ding, Andreas Tolias, and Fabian Sinz. A flow-based latent state generative model of neural population responses to natural images. *Advances in Neural Information Processing Systems*, 34: 15801–15815, 2021.

Mark Bear, Barry Connors, and Michael A Paradiso. *Neuroscience: Exploring the Brain, Enhanced Edition: Exploring the Brain*. Jones & Bartlett Learning, 2020.

James A Bednar and Stuart P Wilson. Cortical maps. *The Neuroscientist*, 22 (6):604–617, 2016.

Lukas Biewald. Experiment tracking with weights and biases. *Software available from wandb. com*, 2:233, 2020.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94(5):3637–3642, 2005.

Daniel A Butts. Data-driven approaches to understanding visual neuron activity. *Annual review of vision science*, 5:451–477, 2019.

Santiago A Cadena, George H Denfield, Edgar Y Walker, Leon A Gatys, Andreas S Tolias, Matthias Bethge, and Alexander S Ecker. Deep convolutional models improve predictions of macaque v1 responses to natural images. *PLoS computational biology*, 15(4):e1006897, 2019.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Alexander S Ecker, Fabian H Sinz, Emmanouil Froudarakis, Paul G Fahey, Santiago A Cadena, Edgar Y Walker, Erick Cobos, Jacob Reimer, Andreas S Tolias, and Matthias Bethge. A rotation-equivariant convolutional neural network model of primary visual cortex. *arXiv preprint arXiv:1809.10504*, 2018.

Eduardo Fernández, Arantxa Alfaro, Cristina Soto-Sánchez, Pablo Gonzalez-Lopez, Antonio M Lozano, Sebastian Peña, Maria Dolores Grima, Alfonso Rodil, Bernardeta Gómez, Xing Chen, et al. Visual percepts evoked with an intracortical 96-channel microelectrode array inserted in human occipital cortex. *The Journal of Clinical Investigation*, 131(23), 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Robbe LT Goris, J Anthony Movshon, and Eero P Simoncelli. Partitioning neuronal variability. *Nature neuroscience*, 17(6):858–865, 2014.

Andreas Heinecke, Jinn Ho, and Wen-Liang Hwang. Refinement and universal approximation via sparsely connected relu convolution nets. *IEEE Signal Processing Letters*, 27:1175–1179, 2020.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466, 1952.

Kevin Kilgore. *Implantable neuroprostheses for restoring function*. Woodhead Publishing, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

David Klindt, Alexander S Ecker, Thomas Euler, and Matthias Bethge. Neural system identification for large populations separating "what" and "where". *Advances in Neural Information Processing Systems*, 30, 2017.

Nikolaus Kriegeskorte. Deep neural networks: a new framework for modelling biological vision and brain information processing. *biorxiv*, page 029876, 2015.

Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5):e0177459, 2017.

Stanley YM Lam, Bertram Emil Shi, and Kwabena A Boahen. Self-organized cortical map formation by guiding connections. In *2005 IEEE International Symposium on Circuits and Systems*, pages 5230–5233. IEEE, 2005.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Philip M Lewis and Jeffrey V Rosenfeld. Electrical stimulation of the brain and the development of cortical visual prostheses: an historical perspective. *Brain research*, 1630:208–224, 2016.

Konstantin-Klemens Lurz, Mohammad Bashiri, Konstantin Willeke, Akshay K Jagadish, Eric Wang, Edgar Y Walker, Santiago A Cadena, Taliah Muhammad, Erick Cobos, Andreas S Tolias, et al. Generalization in data-driven models of primary visual cortex. *bioRxiv*, pages 2020–10, 2021.

Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

Luis Miralles-Pechuán, Dafne Rosso, Fernando Jiménez, and Jose M García. A methodology based on deep learning for advert value calculation in cpm, cpc and cpa networks. *Soft Computing*, 21(3):651–665, 2017.

Rebecca M Mirochnik and John S Pezaris. Contemporary approaches to visual prostheses. *Military Medical Research*, 6(1):1–9, 2019.

NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. URL `https://developer.nvidia.com/cuda-toolkit`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Eftychios A Pnevmatikakis, Daniel Soudry, Yuanjun Gao, Timothy A Machado, Josh Merel, David Pfau, Thomas Reardon, Yu Mu, Clay Lacefield, Weijian Yang, et al. Simultaneous denoising, deconvolution, and demixing of calcium imaging data. *Neuron*, 89(2):285–299, 2016.

Dale Purves. *Neuroscience*. Oxford University Press, 2019.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Fabian Sinz, Alexander S Ecker, Paul Fahey, Edgar Walker, Erick Cobos, Emmanouil Froudarakis, Dimitri Yatsenko, Zachary Pitkow, Jacob Reimer, and Andreas Tolias. Stimulus domain transfer in recurrent models for large scale cortical population prediction on video. *Advances in neural information processing systems*, 31, 2018.

Gordon B Smith, Bettina Hein, David E Whitney, David Fitzpatrick, and Matthias Kaschube. Distributed network interactions and their emergence in developing neocortex. *Nature neuroscience*, 21(11):1600–1608, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Nicholas James Sofroniew, Daniel Flickinger, Jonathan King, and Karel Svoboda. A large field of view two-photon mesoscope with subcellular resolution for in vivo imaging. *elife*, 5:e14472, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in science & engineering*, 13(2):22–30, 2011.

Stephen D Van Hooser, J Alexander F Heimel, Sooyoung Chung, Sacha B Nelson, and Louis J Toth. Orientation selectivity without orientation maps in visual cortex of a highly visual mammal. *Journal of Neuroscience*, 25(1):19–28, 2005.

Edgar Y Walker, Fabian H Sinz, Erick Cobos, Taliah Muhammad, Emmanouil Froudarakis, Paul G Fahey, Alexander S Ecker, Jacob Reimer, Xaq Pitkow, and Andreas S Tolias. Inception loops discover what excites neurons most using deep predictive models. *Nature neuroscience*, 22(12):2060–2065, 2019.

Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018.

Konstantin F Willeke, Paul G Fahey, Mohammad Bashiri, Laura Pede, Max F Burg, Christoph Blessing, Santiago A Cadena, Zhiwei Ding, Konstantin-Klemens Lurz, Kayla Ponder, et al. The sensorium competition on predicting large-scale mouse primary visual cortex activity. *arXiv preprint arXiv:2206.08666*, 2022.

Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020.

List of Figures

List of Tables

List of Abbreviations

DNN Deep neural network

CNN Convolutional neural network

reCNN Rotation-equivariant convolutional neural network

LGN Lateral geniculate nucleus

V1 Primary visual cortex.

val Validation (dataset)

test Test (dataset)

corr Correlation

wrt. with respect to

MLE Most likelihood estimation.

A. Attachments

A.1 reCNN-visual-prosthesis

A repository with all the source code of our solution. The repository can also be found at `https://github.com/mpicek/reCNN_visual_prosthesis"`.

A.2 csng-dl-docker-image

A repository with a Docker image for running the training environment. Also available online at `https://github.com/mpicek/csng_dl_docker_image`.