Modeling early visual cortex using neural network models with recurrent circuits

Yimeng Zhang

CMU-CS-21-105
March 2021

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Tai Sing Lee, Chair
Abhinav Gupta
Robert E. Kass
Alan L. Yuille, Johns Hopkins University

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

This research was sponsored by the National Institute of Health under award number R01EY022247, the Department of Interior under award number D16PC00007, and the National Science Foundation under award numbers IIS1320651 and IIS1816568.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: computational neuroscience, computational vision, neural data fitting, early visual cortex, primary visual cortex, convolutional neural networks, recurrent convolutional neural networks, Boltzmann machines

For Cheng, Xiaoli, and Jane

Abstract

In this thesis, I develop, test, and understand neural network models of recurrent circuits for V1 and early visual areas in general in a series of three studies. In the first study, I trained a Boltzmann machine using 3D natural scene data and established the consistency of the learned connections of such a recurrent network to the functional connectivity of neurons measured in neurophysiological experiments, and thus showing that the recurrent connectivity of the brain reflects statistical priors of the natural scenes. In the second study, I compared feed-forward convolutional neural networks (CNNs) with other popular models for predicting V1 neural responses and isolated the key components underlying the superior performance of CNN models. In the third study, I introduced recurrent circuits to the CNN and showed that recurrent models provided better predictive performance and were more data efficient compared to feed-forward models of comparable configurations. The learned recurrent models could reproduce a variety of contextual modulation effects observed in the visual cortex. To understand the computational advantage of the recurrent models, I proposed a new conceptualization of the recurrent network as a multi-path ensemble model and I established that compared to feed-forward models, multi-path ensembles as implemented by recurrent models can be more flexible and data efficient in learning and approximating complex computations as those in the brain.

Acknowledgments

This work would not have been possible without my advisor Dr. Tai Sing Lee, who lent me enormous help and guidance throughout the course of the adventure. I'd also like to thank everyone in the thesis committee that gave me invaluable advice to help sharpen and polish my work.

I'd like to thank everyone I met in the lab, especially Ge Huang, Mingmin Zhao, Chengxu Zhuang, Zhihao Li, Xingyu Lin, Hao Wang, Esha Uboweja, Tiancheng Zhi, Faisal Baqai, Bowen Xiao, Siming Yan, Yue Xu, Ziniu Wu, Xuyang Fang, and Hal Rockwell; each of them inspired me to try some new idea from time to time during this long journey.

To me, good computational research depends on not only brilliant ideas, but also solid execution supported by powerful and reliable computational infrastructure. I thank David Pane and his team for maintaining all the computing resources around the clock so that I could execute all the ideas explored in this work efficiently.

Carnegie Mellon is a unique place with all kinds of talents whom I can make friends with. I'd like to thank Pengcheng Zhou, Xiao Zhou, Zongge Liu, Cheng Xue, and Wen-Hao Zhang, for their encouraging life conversations and uplifting spirits.

While being cold in winter, Pittsburgh always feels a warm and cozy place to me, due to all the fond memories I have with friends here and there in the steel city. I'd like to give special thanks to Wang Zhang, Pengcheng Zhou (again), and Yunsheng Cao, for all the unforgettable movie nights with homemade delicacies.

Lastly, I'd like to thank my parents, who have been understanding since the beginning of this long, winding endeavor, and Jane, who has been supportive and caring since the very first day we met.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The early visual cortex [97, 118, 170], which consists of primary visual cortex (V1), secondary visual cortex (V2), and sometimes V3 as well, participates in the first stages of visual processing in the brain.

Receiving visual sensory input ultimately from the retina, V1 neurons are sensitive to various stimulus attributes such as orientation, spatial frequency, and direction; they are traditionally classified as simple and complex cells based on their responses to drifting visual gratings [63, 64, 65]. Traditionally, simple and complex V1 neurons are modeled by linear-nonlinear (LN) models [52] and energy models [1], respectively. By construction, these standard V1 models (LN models and energy models) only respond to stimulus change with in neurons' classical receptive fields or CRFs [49], which are classically defined as regions in the visual space where the presence or absence of small, impulse like stimuli (bright or dark dots) can cause change in neural response.

Anatomically adjacent to V1, V2 receives strong feed-forward inputs from V1 neurons [165] and contains many orientation selective neurons, resembling its V1 input [37, 66]. On the other hand, many V2 neurons respond well to complex stimuli such as angles and curves [4, 53, 68]. Existing V2 computational studies mostly model V2 neurons as linear combinations of V1 neurons [96, 120, 176], based on existing V1 models.

The standard V1 models described above have been successful in explaining V1 response to relatively simple stimuli, such as bars, edges and gratings [142]. However, they cannot explain satisfactorily neural responses to more complex stimuli such as complex shapes and natural images [13, 25, 54, 87, 160, 171], under which V1 neurons exhibit complex nonlinear response properties; these properties not predicted by standard models are collectively called *non-classical receptive field (nCRF) effects* [185] in this document. When natural stimuli are used, V1's nCRF effects result in a significant portion (no less than $50\,\%$) of variance in neural responses left unexplained by most existing models, including those state-of-the-art ones based on feed-forward neural networks for predicting V1 data [13, 25, 81, 87]. Given the inadequacy of its component V1 models, existing V2 models leave a higher portion of neural responses unexplained compared to V1 models [176].

One possible reason for the failure of existing models in terms of explaining neural responses to natural stimuli is the lack of recurrent connections in these models. In the brain, it is well known that there are local horizontal recurrent connections between neurons in the same area [73] and long-range feedback recurrent connections between neurons in different areas [32]. By some count [27], more than 90% of the excitatory synapses and virtually all of the inhibitory ones in V1 come from recurrent connections. These connections greatly contribute to the complexity of the visual system, and may be essential for the success of the visual systems in reality; for example, there are evidences that recurrent connections are crucial for object recognition under noise, clutter, and occlusion [121, 133, 154].

One possible way to study the potential role of recurrent connections in nCRF effects and other phenomena in V1 is to model them using neural networks. In recent years, neural network models have been used to study various visual areas with great success [13, 78, 81, 89, 179, 184]. As a consequence of their current success and biological realism relative to other computational models, neural network models provide a viable tool for exploring computational (and, to a lesser extent, mechanistic) mechanisms of V1 and early visual areas in general. Furthermore, as neural network models in general allow end-to-end training more easily than other computational models, V1 models based on neural networks can be trained on data sets containing large numbers of neurons and complex, natural stimuli to test the validity of these models in a more realistic setting. Recent studies have begun to explore the benefits of recurrent connections in many machine learning settings [23, 106, 108, 116]; however, the role recurrent connections play in the primary visual cortex and early visual areas in general remains unexplained. While there have been many studies on modeling various neural response properties (surround suppression, end-stopping, etc.) thought to related to recurrent circuits [18, 19, 20, 134, 156, 157, 158, 180, 185], these models' predictive power on arbitrary input stimuli are either unknown or worse compared to deep feed-forward networks that have recently become popular in the visual neuroscience community for predicting neural responses to arbitrary stimuli [13, 78, 81, 89, 105, 179].

1.2 Summary of the work

This thesis tries to develop, test, and understand neural network models of recurrent circuits for V1 and early visual areas in general. To achieve the goal, I've conducted three studies as summarized as follows.

1.2.1 Relating recurrent connections to natural scene statistics

In the first study (Chapter 2) [182], I have demonstrated that the Boltzmann machine, which is inspired by neuroscience but often dismissed to be too abstract and different from the real brain, is in fact a useful and viable model for conceptualizing certain recurrent computations in the primary visual cortex. In particular, I attempted to learn horizontal recurrent connections between V1 disparity-tuned neurons by learning from 3D natural scene data using a Boltzmann machine; the learned connectivity patterns were consistent with connectivity constraints in stereopsis models [110, 147], and simulated neurophysiological experiments on the model were consistent with neurophysiological data in terms of functional connectivities among disparity-tuned neurons in V1 [145].

1.2.2 Feed-forward circuit characterization of V1 neurons using convolutional neural networks

In the second study (Chapter 3) [184], I systematically evaluated the relative merits of different (feedforward) CNN components in the context of modeling V1 neurons. The study demonstrated that key components of the CNN (convolution, thresholding nonlinearity, and pooling) contributed to its superior performance in explaining V1 responses to complex stimuli [160] and these key components are consistent with previous V1 modeling and neurophysiology studies. While not directly related to recurrent circuits, it has demonstrated that predicting neural responses to natural and complex stimuli accurately is a useful objective metric for identifying neural network models with high correspondence with biological reality; in addition, it has shown the usefulness of various ablation, dissection, and visualization methods for comparing and understanding neural network models of different architectures. The infrastructure and analysis tools developed in the study to train and analyze tens of thousands of models with different parameters will be heavily used in the third and final study.

1.2.3 Modeling neural responses of early visual areas using recurrent convolutional neural networks

In the third study (Chapter 4), I tried to answer two questions regarding recurrent network models in modeling early visual areas. Experimentally, I wanted to know whether deep neural networks with recurrent circuits can provide a better model for predicting neural responses in the early visual areas; theoretically, I wanted to know why recurrent models perform better from either a computational or a biological perspective, if the answer to the first question is positive.

To answer the first question, I trained tens of thousands of recurrent models with different hyperparameters and under different amounts of training data using different data sets. I found that recurrent models could explain neural responses of early visual areas better than typical feed-forward models with matched hyperparameters and model sizes, especially when there was less training data.

To answer the second question, I developed a novel method to reformulate a recurrent model as an ensemble of feed-forward models. Our novel method is based on the hypothesis that the advantage of the recurrent model rests on the ensemble of multiple feed-forward paths embedded in the recurrent computation and such multitude of paths makes the recurrent model more flexible compared to the feed-forward model. By studying recurrent models and feed-forward models via their corresponding multi-path ensembles, we found that the recurrent model outperformed the feed-forward one due to the former's compact and implicit multi-path ensemble that allows approximating the complex function underlying recurrent biological circuits with efficiency. In addition, we found that the performance differences among the recurrent models we explored were highly correlated with the differences in their multi-path ensembles; in particular, models with more relative weights on shorter paths tended to perform better than models with more relative weights on longer paths.

This study establishes that the recurrent model performs better than the purely feed-forward model for predicting neural responses in the early visual areas, complementing previous studies on feed-forward models [13, 78, 81, 179] and consistent with very recent studies on recurrent ones [77, 155]. The most interesting contribution of this study is to establish that the superiority of the recurrent model for neural prediction can be attributed to the implicit and compact multi-path ensemble inside the model, and that a balance of different paths in the ensemble is necessary for the model to achieve the best performance. This work provides new understanding on the computational rationales and advantages of recurrent circuits that are ubiquitous in biological systems [32].

Chapter 2

Relating functional connectivity in V1 neural circuits and 3D natural scenes using Boltzmann machines

Bayesian theory has provided a compelling conceptualization for perceptual inference in the brain. Central to Bayesian inference is the notion of statistical priors. To understand the neural mechanisms of Bayesian inference, we need to understand the neural representation of statistical regularities in the natural environment. In this study, we investigated empirically how statistical regularities in natural 3D scenes are represented in the functional connectivity of disparity-tuned neurons in the primary visual cortex of primates. We applied a Boltzmann machine model to learn from 3D natural scenes, and found that the units in the model exhibited cooperative and competitive interactions, forming a "disparity association field", analogous to the contour association field. The cooperative and competitive interactions in the disparity association field are consistent with constraints of computational models for stereo matching. In addition, we simulated neurophysiological experiments on the model, and found the results to be consistent with neurophysiological data in terms of the functional connectivity measurements between disparity-tuned neurons in the macaque primary visual cortex. These findings demonstrate that there is a relationship between the functional connectivity observed in the visual cortex and the statistics of natural scenes. They also suggest that the Boltzmann machine can be a viable model for conceptualizing computations in the visual cortex and, as such, can be used to predict neural circuits in the visual cortex from natural scene statistics.

2.1 Introduction

Natural scenes contain significant ambiguity. To resolve ambiguities and obtain a stable 3D percept of the world, the visual system (as well as the whole brain) must perform inference, integrating current sensory data with prior knowledge of the world formulated from past experience. Therefore, (Bayesian) inference has long been proposed as a fundamental computational principle of the brain [82, 173]. In this work, we attempt to address one of the key questions for understanding Bayesian inference in the brain, in the context of the primary visual cortex (V1):

how might an internal model of natural scenes—the Bayesian prior—be encoded in the brain?

To support visual inference, an internal representation of the visual scenes requires encoding both the statistical regularities of the boundaries and of the surfaces themselves. There have been studies suggesting that the neural circuits in the primary visual cortex (V1) encode contour priors in the form of the contour association field [8, 31, 33, 42, 71, 103, 114, 145, 147]. Recent neurophysiological evidence suggests that disparity-tuned neurons in the primary visual cortex might form a recurrent network for stereo processing [145, 147]. This network encodes the statistical correlation of disparity signals in natural scenes, complementing the contour association field, and is referred to as the disparity association field. However, the neural mechanisms by which statistical priors of boundaries and surfaces from the environment can be learned are not well understood.

We hypothesize that the empirically observed neural connectivity between disparity-tuned neurons in V1 can be predicted from 3D natural scenes using a Boltzmann machine. To test this hypothesis, we fitted a Boltzmann machine neural network model [56] to disparity signals derived from 3D natural scene data, and found that 1) learned parameters in the model were consistent with connectivity constraints in stereopsis models [110, 147]; 2) the model was consistent with neurophysiological data in terms of functional connectivities among disparity-tuned neurons in V1 [145]. The results provide further evidence in support of the notion of the disparity association field, and demonstrate that the Boltzmann machine is a viable model for describing cortical computation in the sense that they can be used to predict functional neural circuitry in the visual cortex.

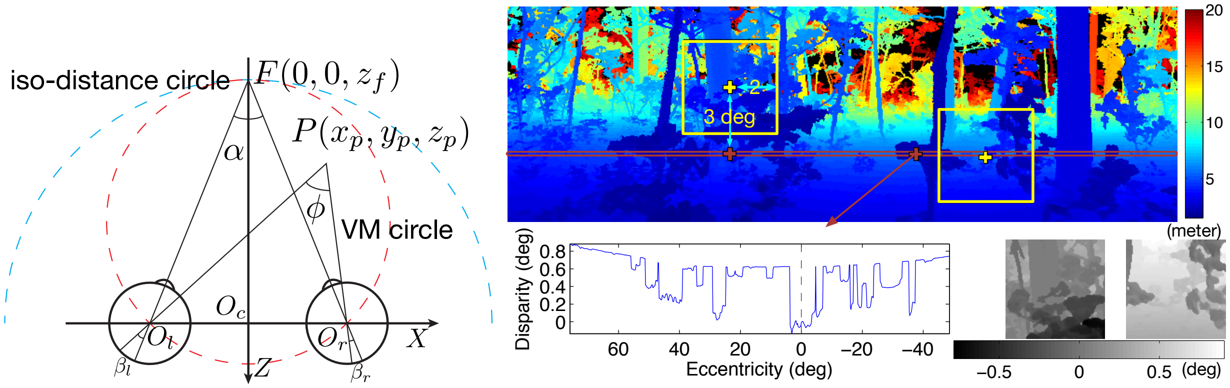
The study is organized as follows. In Section 2.2, we describe the 3D natural scene data and the Boltzmann machine model, as well as the neurophysiological experiments for measuring functional connectivities between pairs of neurons. In Section 2.3, we compare the trained Boltzmann machine with computational models and neurophysiological data. In Section 2.4, we discuss the potential implications of this model and its limitations.

2.2 Methods

2.2.1 3D scene data

We trained a Boltzmann machine to model the disparity signals over a small visual field. These signals were derived from the Brown Range Image Database [62]. A total of 200K disparity image patches with a 2° half-width were extracted from 172 images (54 forest, 49 interior, 69 residential). The images in the Brown data set were captured by a scanner with range at $2\,\mathrm{m}$ to $200\,\mathrm{m}$, and image resolutions were approximately 5 pixels per degree of visual angle.

Disparity image patches were generated from each range image as follows (Figure 2.1b). A random point in the range image was chosen as the fixation point. Given the fixation point, the disparities at its surrounding pixels were computed using the method in Liu et al. [107] (see Section 2.2.1.1 for detail). Finally, a disparity image patch with a 2° half-width was extracted 3° away from the fixation point. This eccentricity was chosen to roughly match the typical receptive field locations of recorded V1 neurons in our earlier neurophysiological experiments.



(a)

(b)

Figure 2.1: **(a)** Diagram for calculating disparity. Adapted from Liu et al. [107]. See Eqs. (2.1)-(2.3) for detail. **(b)** One sample range image from the Brown data set (upper) with disparity values along one line in it (lower left), and two extracted disparity patches (lower right). In the upper image: red crosses, fixations points for two patches; yellow crosses, center of patches; red long rectangle, the row shown disparities. Patches were $3°$ away from fixation and had a half-width of $2°$.

2.2.1.1 Disparity computation

We used an optical model of the primate eye following Liu et al. [107] to compute disparity. In this model (Figure 2.1a), each eye is approximated as a perfect sphere centered at its nodal point, and inter-pupillary distance is assumed to be $0.038\,\mathrm{m}$ with nodal points at $(-0.019, 0, 0)$ and $(0.019, 0, 0)$ as in monkey physiology.

Consider some fixation point $F = (x_f, y_f, z_f)$. Let $O_c = (0, 0, 0)$ be the midpoint between the two eyes. We assume all observations are directed along the $-z$ axis, or $x_f = y_f = 0$. The distance from O_c to F is then just z_f. The horizontal disparity, d, of an arbitrary point $P = (x_p, y_p, z_p)$, is given by

$$d = \beta_r - \beta_l = \alpha - \phi, \tag{2.1}$$

$$\alpha = 2\,\mathrm{atan}(-0.019/z_f), \tag{2.2}$$

$$\phi = \mathrm{atan}\left(\frac{-x_p - 0.019}{z_p}\right) - \mathrm{atan}\left(\frac{-x_p + 0.019}{z_p}\right). \tag{2.3}$$

We made the simplifying assumption that fixations occur at any point in the scene with uniform probability. This assumption is supported by Liu et al. [107], which shows that random fixations roughly emulate the statistics of fixation, at least in natural scenes. This assumption should not affect the basic conclusion of our results.

2.2.2 Boltzmann machines

2.2.2.1 Interaction among neurons modeled by Boltzmann machines

The extracted disparity image patches reflect the prior of disparity signals in the natural scene, and we modeled this prior by fitting a Boltzmann machine to the patches. Boltzmann machines [56] are a class of stochastic recurrent neural network models that can learn internal representations to explain or generate the distribution of the input data, using pairwise connectivity between units to encode the structures of the input data. Boltzmann machines are also a type of Markov random fields, which are widely used in computer vision for solving a variety of early vision problems such as surface interpolation and stereo inference [6, 43, 85, 162]. We hypothesize that Boltzmann machines are a viable computational model for understanding the circuitry of the visual cortex, and we will examine if they can explain interactions among neurons in other computational and neurophysiological studies [110, 145, 147]. Specifically, the interaction terms $\vec{\boldsymbol{\beta}}$ (Eq. (2.4)) in our Boltzmann machine model were compared with existing computational models in Section 2.3.2, and neurophysiological experiments were simulated on the model (Section 2.2.2.2) to compare it with neural data in Section 2.3.3.

The units in our Boltzmann machine model (Figure 2.2a) are organized into a hidden layer and a visible layer, arranged in a spatial 5 by 5 grid of "hypercolumns" (in total $C = 25$ columns). Each hypercolumn has a bank of $M = 16$ visible units that encode the disparity input, and a bank of 16 corresponding hidden units $\vec{\boldsymbol{h}}$, all sharing the same spatial receptive field location. The $N = MC = 400$ hidden units are fully connected, each of them driven by its corresponding input visible unit. The collective spiking activity at each bank of visible units encodes the disparity signal at the corresponding hypercolumn.

This model is formally expressed as a probability distribution over hidden and visible units:

$$P(\vec{\boldsymbol{h}}, \vec{\boldsymbol{v}}; \vec{\boldsymbol{\alpha}}, \vec{\boldsymbol{\beta}}, \vec{\boldsymbol{\gamma}}, \vec{\boldsymbol{\lambda}}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{N} \alpha_i h_i + \sum_{i<j} \beta_{i,j} h_i h_j + \sum_{i=1}^{N} \lambda_i h_i v_i + \sum_{i=1}^{N} \gamma_i v_i \right). \tag{2.4}$$

In Eq. (2.4), $\vec{\boldsymbol{h}}$ and $\vec{\boldsymbol{v}}$ are binary vectors whose distributions are to be captured by the model, representing spiking activities of hidden and visible units. The other model parameters capture the distributions of $\vec{\boldsymbol{h}}$ and $\vec{\boldsymbol{v}}$, as well as their interactions. Specifically, $\vec{\boldsymbol{\alpha}}$ and $\vec{\boldsymbol{\gamma}}$ capture the baseline firing rates of hidden and visible units, $\vec{\boldsymbol{\beta}}$ models the pairwise lateral interactions among hidden units, and $\vec{\boldsymbol{\lambda}}$ models the interactions between hidden and visible units. Z is a normalization constant.

This Boltzmann machine was fitted by finding parameters $\vec{\boldsymbol{\alpha}}$, $\vec{\boldsymbol{\beta}}$, $\vec{\boldsymbol{\gamma}}$, and $\vec{\boldsymbol{\lambda}}$ that maximize the probability of the model for generating the spike patterns $\vec{\boldsymbol{v}}$, corresponding to the disparity signals in the extracted patches. Formally, the following log likelihood was maximized:

$$L(\vec{\boldsymbol{\alpha}}, \vec{\boldsymbol{\beta}}, \vec{\boldsymbol{\gamma}}, \vec{\boldsymbol{\lambda}}) = \sum_{i=1}^{T} \log P(\vec{\boldsymbol{v}}^{(i)}; \vec{\boldsymbol{\alpha}}, \vec{\boldsymbol{\beta}}, \vec{\boldsymbol{\gamma}}, \vec{\boldsymbol{\lambda}}) = \sum_{i=1}^{T} \log \sum_{j=1}^{2^N} P(\vec{\boldsymbol{h}}^{(j)}, \vec{\boldsymbol{v}}^{(i)}; \vec{\boldsymbol{\alpha}}, \vec{\boldsymbol{\beta}}, \vec{\boldsymbol{\gamma}}, \vec{\boldsymbol{\lambda}}). \tag{2.5}$$

In Eq. (2.5), $\vec{\boldsymbol{v}}^{(i)}$'s are T binary spike patterns of the visible units converted from the disparity signals based on the tuning curves of the visible units (see Figure 2.3 and Section 2.2.2.1.1). The likelihood of observing $\vec{\boldsymbol{v}}^{(i)}$ is computed as the sum of $P(\vec{\boldsymbol{h}}^{(j)}, \vec{\boldsymbol{v}}^{(i)})$ over all possible 2^N hidden unit patterns, which is the marginal probability for the model to generate $\vec{\boldsymbol{v}}^{(i)}$, regardless of the hidden units. Finally, the log probability for the model to generate all the input spike patterns due to the disparity signals is computed as the sum of log probabilities for generating each particular spike pattern $\vec{\boldsymbol{v}}^{(i)}$. The model was trained using contrastive divergence mean field learning [175]. See Section 2.2.2.3 and Welling and Hinton [175] for more detail.

2.2.2.1.1 Conversion of disparity signals into binary spike patterns From each disparity image patch i, disparity values $s_1^i, s_2^i, \dots, s_{C=25}^i$ corresponding to the locations of the 25 hypercolumns were extracted, and the model was fitted to explain these disparity values across all patches. Disparity signals are real-valued, and must be converted into binary spike patterns, which can be considered as the spiking activities of the bottom-up input to V1 neurons. Following the approach of Ganguli and Simoncelli [41], we derived a set of $M = 16$ tuning curves for visible units (same for all the hypercolumns, Figure 2.2c) according to the distribution $P(s)$ of extracted disparity values from all patches (Figure 2.2b). Each disparity value was converted to the mean firing rates of $M = 16$ visible units based on their tuning curves.

Given the above derived tuning curves, for each patch, we first converted the $C = 25$ disparity values into the mean firing rates of the $N = 400$ visible units. Then for each of these N units, a spike train of $200\,\mathrm{ms}$ was generated based on its mean firing rate using an independent homogeneous Poisson process, and the whole spike train was partitioned into 20 bins of $10\,\mathrm{ms}$[1]. A “1” was assigned to a bin of a unit if there were one or more spikes for that unit within that time bin; otherwise, a “0” was assigned. The whole generation process (for one disparity value) is schematically shown in Figure 2.3.

2.2.2.2 Simulation of neurophysiological experiments on the model

With the trained Boltzmann machine, we can simulate the neurophysiological experiments by providing the visible units $\vec{\boldsymbol{v}}$ with specific experimental stimuli (Section 2.2.3.1), and collecting the model response as binary spiking patterns of hidden units $\vec{\boldsymbol{h}}$. Because a Boltzmann machine models the joint distribution of all hidden and visible units, we can compute the model response by sampling from the conditional distribution of hidden units given the visible units:

$$P(\vec{\boldsymbol{h}} \mid \vec{\boldsymbol{v}}; \vec{\boldsymbol{\alpha}}, \vec{\boldsymbol{\beta}}, \vec{\gamma}, \vec{\boldsymbol{\lambda}}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{N} (\alpha_i + \lambda_i v_i) h_i + \sum_{i<j} \beta_{i,j} h_i h_j \right). \tag{2.6}$$

After generating hidden unit activities $\vec{\boldsymbol{h}}$ by drawing samples from Eq. (2.6) (see Section 2.2.2.2.1 for detail), we compared $\vec{\boldsymbol{h}}$ with neural data (Section 2.3.3), in terms of functional connectivity using methods described in Section 2.2.3.2.

[1] Our implementation was written in terms of bins, with no notion of the physical duration of each bin. We arbitrarily assumed each bin to be of $10\,\mathrm{ms}$, for easier comparison with neurophysiological data and other studies based on Ising models (a type of Boltzmann machines).

2.2.2.2.1 Sampling of hidden unit activities given disparity stimulus Given the (real-valued) disparity values of the stimulus at $C = 25$ hypercolumns, we first converted them into mean firing rates for all input visible units according to the tuning curves. Then we obtained each sample of $\vec{\boldsymbol{h}}$ (a N-dimensional binary vector) in the following MCMC fashion [86].

1. generate a $\vec{\boldsymbol{v}}$ from the Poisson process described in Section 2.2.2.1.1, based on mean firing rates of visible units.
2. initialize $\vec{\boldsymbol{h}}$ randomly, run Gibbs sampling for one step[2] based on Eq. (2.6).
3. collect the current $\vec{\boldsymbol{h}}$ as a sample.
4. start over from 1), but when in 2), initialize $\vec{\boldsymbol{h}}$ with the previous sample.

$20\,000$ samples were generated for each stimulus, and every contiguous 100 samples were regarded as the model response in a trial, with 100 samples between trials, resulting in 100 trials of 100 samples per stimulus[3]. In addition, before collecting the first sample, we performed an additional 100 Gibbs sampling steps, as "burn-in".

2.2.2.3 More implementation details

For results shown in Section 2.3, $48\,960$ disparity patches[4] were extracted from the Brown data set to train the Boltzmann machine model, with distance between nearby hypercolumns set to 1° (Figure 2.2a). Data was taken in mini-batches of size 1000, and training took 1000 epochs. A learning rate of 1×10^{-2} was used for learning bias terms $\vec{\boldsymbol{\alpha}}$ and $\vec{\boldsymbol{\gamma}}$, and half of that for learning lateral connections $\vec{\boldsymbol{\beta}}$. Multiplicative weight decay of 1×10^{-2} for $\vec{\boldsymbol{\beta}}$ multiplied by the learning rate was used, and a momentum factor of 0.5 for first five epochs and one of 0.9 for the rest were employed. Five iterations of mean field updates were used per iteration, with damping parameter set to 0.2. Because $\vec{\boldsymbol{\alpha}}$, $\vec{\boldsymbol{\gamma}}$, and $\vec{\boldsymbol{\lambda}}$ offer too many degrees of freedom for the model fitting, we fixed $\vec{\boldsymbol{\lambda}}$ to be positive, all elements equal to 0.5 during training. This encouraged the resulting $\vec{\boldsymbol{\alpha}}$ and $\vec{\boldsymbol{\gamma}}$ to be negative, and the hidden units to share the same preferred disparities as their visible input units. If we did not constrain $\vec{\boldsymbol{\lambda}}$, we found that the learned $\vec{\boldsymbol{\alpha}}$ and $\vec{\boldsymbol{\gamma}}$ were positive, and the tuning curves of hidden units would be inverted from those of visible units, both counter-intuitive.

[2] Here one step of Gibbs sampling is defined as in Koller and Friedman [86], that is, given the initial state of all hidden units $h_1, h_2, \ldots, h_N$, we randomly choose one unit h_i, update it based on $P(h_i \mid \vec{\boldsymbol{h}}_{-i})$ (where $-i$ means all but i), and do this update sequentially for all N units. These N updates are collectively referred to as one step.

[3] Given all $20\,000$ samples, samples 1–100, 201–300, 401–500, ..., $19\,801$–$19\,900$ were collected as trials, and samples between them (samples 101–200, 301–400, 501–600, ...) were discarded. This yielded 100 samples per trial, and 100 trials per stimulus.

[4] These patches were from 49 interior images in the Brown data set, with 1000 patches per image. A total of 40 patches were dropped because they had missing range data (thus disparity signals) for some hypercolumns. Other patches from different parts of the data set such as forest scenes were also tried, with empirically similar results, also shown in the Results section.

2.2.3 Neurophysiological experiments

2.2.3.1 Neurophysiological data

We compared the predictions from the trained Boltzmann machine with observations about the neural circuitry that we have reported in previous studies, based on direct pairwise measurements of neuronal spiking activities [145], and a recurrent neural circuit model that predicts neural responses better than the feed-forward energy model [147].

In these earlier experiments, we analyzed neural data recorded using multi-electrode recording techniques from neurons in the primary visual cortex of three awake behaving macaque monkeys. For monkeys D and F, we used 4-8 tungsten in epoxy or glass microelectrodes [145], while for monkey I we recorded from neurons using a chronically implanted multielectrode array with 96 channels [146, 147]. The experimental protocols for these studies were approved by the Institutional Animal Care and Use Committee of Carnegie Mellon University and in accordance with Public Health Service guidelines for the care and use of laboratory animals.

Stimuli were dynamic random dot stereograms (DRDS) presented for one second per trial while the monkey performed a fixation task. Each DRDS defines a uniform fronto-parallel depth plane (i.e. uniform disparity) inside a $3.5°$ visual angle aperture window over the receptive fields of the neurons being recorded. These stimuli were standard stimuli used to assess disparity tuning of the neurons and were effective in driving disparity-tuned neurons. The dynamic random dot stereogram (DRDS) was composed of 25-percent black and white dots on a mean gray background with a refresh rate of $12\,\mathrm{Hz}$ for dot patterns (monitor refresh rate was $120\,\mathrm{Hz}$) at 11 disparities (±0.94°; ±0.658°; ±282°; ±0.188°; ±0.094°; 0°). Further details about the neurophysiological experimental procedures are described in our previous works [145, 146, 147].

2.2.3.2 Neurophysiological measures of interaction

We measured the functional connectivity between pairs of neurons using a cross-correlation histogram (CCH) measure based on standard methods [2, 145]. The probabilities of joint spike occurrences beyond chance at all possible lag times and all times from stimulus onset were computed by measuring the observed probability of joint occurrences and subtracting the expected joint occurrences, which was the outer product of the peristimulus time histograms:

$$C_{xy}(t_1,t_2) = \langle x(t_1), y(t_2)\rangle - \langle x(t_1)\rangle\langle y(t_2)\rangle, \tag{2.7}$$

where $x(t), y(t)$ were the spike trains of the two neurons respectively.

The expected joint occurrences $\langle x(t_1)\rangle\langle y(t_2)\rangle$ account for the stimulus-related response correlation assuming neurons are independent. They were corrected for trial-to-trial changes in the firing rate to remove potential slow sources of correlation that can lead to apparent fast sources of correlation sometimes referred to as an excitability correction [10, 44, 145, 169]. This two-dimensional cross covariance histogram was then normalized by the square root of the product of the auto-covariance histograms for the two neurons:

$$D_{xy}(t_1,t_2) = \frac{C_{xy}(t_1,t_2)}{\sqrt{C_{xx}(t_1,t_1)C_{yy}(t_2,t_2)}}. \tag{2.8}$$

This produced a two-dimensional histogram of Pearson's correlation coefficients referred to as the normalized cross-covariance histogram or the normalized joint poststimulus histogram. We average this histogram across diagonals to produce a cross-correlation histogram with respect to lag times between pairs of neurons. We then computed the variance of our estimates by bootstrapping with respect to trials [30, 145, 168]. The correlation measurements were the areas under the half-height of the peaks of these cross-correlation histograms for pairs of neurons with a central peak (within $10\,\mathrm{ms}$ of 0 lag time) more than three standard deviations above or below the cross-correlation histogram from $175\,\mathrm{ms}$ to $375\,\mathrm{ms}$ lag time and for pairs of neurons where both neurons had significant disparity tuning (1-way ANOVA, $p < 0.01$) [145]. For the neurophysiological results shown in this work (Figures 2.8b,d,f), the input stimulus for each pair of neurons was set to the the disparity at which the point-wise product of two neurons' tuning curves was at its maximum [145].

We also computed CCH measures for hidden unit activities generated from the simulation of the trained Boltzmann machine model (Section 2.2.2.2). To compute CCH for each pair of hidden units in the model, we set the input stimulus (equi-disparity stimulus, with the same disparity value at all hypercolumns) to be at the mean of the preferred disparities of their corresponding visible units[5]. After collecting the samples (see Section 2.2.2.2.1 for detail), we computed the CCH measure following the method described above[6], but using only the peak of CCH as the CCH measure of this pair of units, since there was no synaptic delay issue in our Boltzmann machine model, given our simulation method.

[5] We did this because the preferred disparities of hidden units were actually those of their input visible units. See Figure 2.4a for detail.

[6] Since our model had $N = 400$ hidden units, we initially computed the CCH for all $400 \times 399/2 = 79\,800$ pairs. Then we kept pairs whose CCH's peak within $50\,\mathrm{ms}$ (±5 bins) of 0 lag time was more than 1.5 standard deviations above or below the CCH from $300\,\mathrm{ms}$ to $600\,\mathrm{ms}$ (±30 to ±60 bins). We further removed pairs where at least one unit responded to the input stimulus with firing rate less than half of its peak response relative to its minimum response over all tested stimuli, resulting in $23\,048$ pairs shown in Figure 2.8.

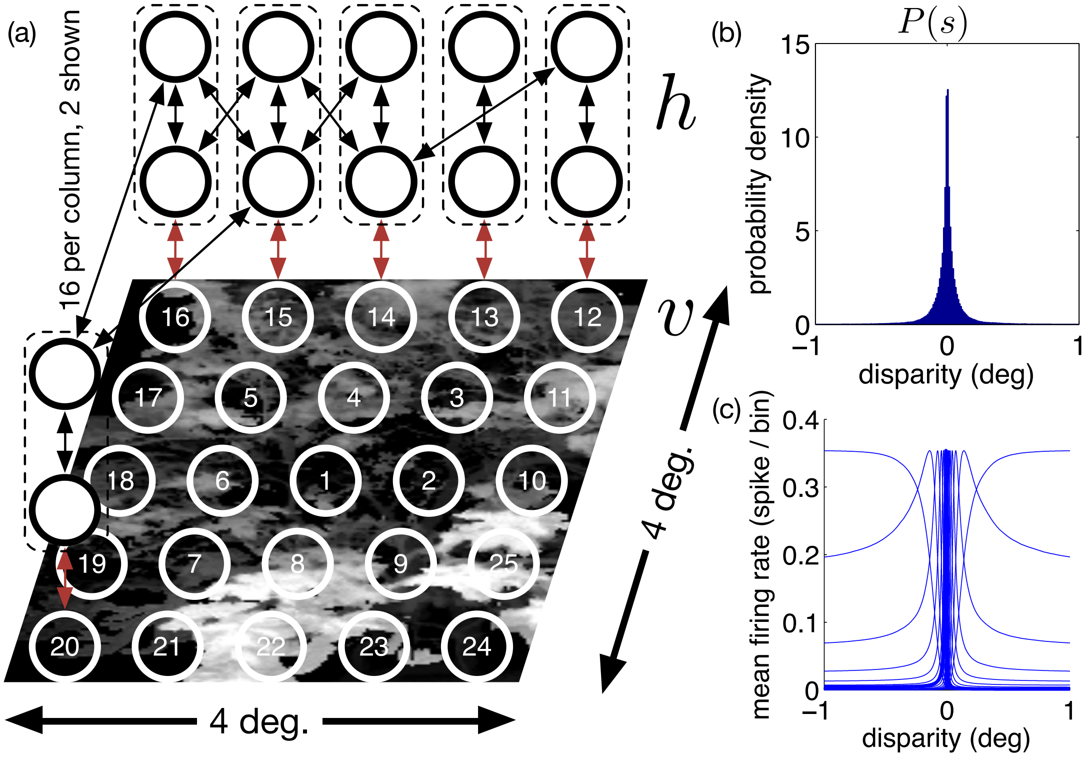
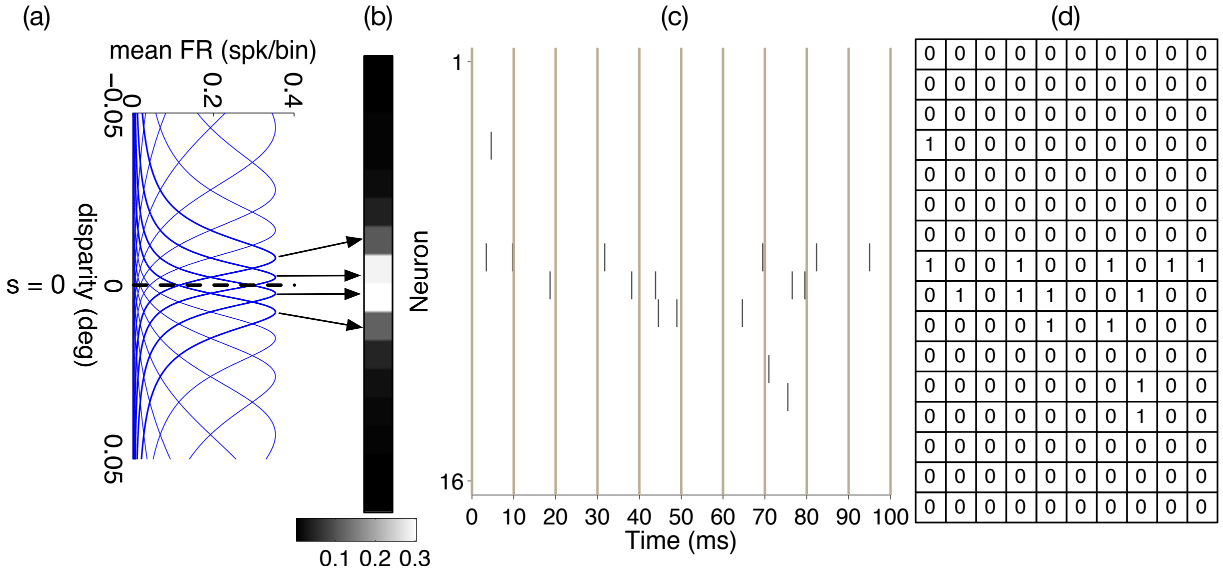


Figure 2.2: Schematic of our Boltzmann machine model (a), distribution of extracted disparity values $P(s)$ (b), and derived tuning curves of input visible units, with one curve highlighted (c). **(a)** 25 "hypercolumns" laid in a 5 by 5 grid covering a $4°$ by $4°$ patch, with hidden units ($\vec{\boldsymbol{h}}$, black outline) in the same column grouped in dashed box. Each hidden unit has connections (black) to all other ones, and one connection (red) to its own visible unit ($\vec{\boldsymbol{v}}$, white outline). At most two hidden units and one visible unit drawn per column, with many connections missing for clarity. Columns are numbered for reference in Section 2.3. **(b)** The distribution of extracted disparity values was sharp, and peaked at zero. **(c)** Tuning curves of $\vec{\boldsymbol{v}}$ were derived based on Ganguli and Simoncelli [41] with the following details: "baseline" curve was a t distribution with d.o.f. $\nu = 2$, total expected firing rate (R in Ganguli and Simoncelli [41]) was unity, and "infomax" optimization criterion was used. Only tuning curves between $-1°$ and $1°$ are shown for clarity. Given the sharp distribution of disparity values, the theory in Ganguli and Simoncelli [41] made the tuning curves at large disparities different from those close to zero. Instead of Gaussian-like (see Figure 2.3a for a zoom-in view of tuning curves close to zero), the tuning curves at the two ends of the input distribution were relatively flat at large (positive/negative) disparities, and dropped to zero near zero disparity. Interestingly, these were very similar to the near-tuned and far-tuned neurons in Poggio and Fischer [129] and Poggio et al. [130]. We also tried Gaussian distribution as the "baseline" curve, but that gave much sharper tuning curves and less co-activation between dissimilar units, which resulted in a less biologically realistic training result.



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1	1
0	1	0	1	1	0	0	1	0	0
0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 2.3: Generation of training data for one disparity value. Given one disparity value **(a)** (in this case $s = 0$), we transformed it into $M = 16$ mean firing rates (b) using tuning curves (between **(a)** and **(b)**), generated spike trains **(c)**, and binned it into a binary matrix **(d)** as training data to the Boltzmann machine.

2.3 Results

We mainly compared the model with existing computational models in terms of connectivity constraints (Section 2.3.2), and neurophysiological data in terms of functional connectivities (Section 2.3.3). The model showed qualitative agreement in both aspects. In the following comparisons, the hidden units correspond to the disparity-tuned V1 neurons, likely realized in terms of the complex cells in the superficial layer of V1 where there are extensive horizontal axonal collaterals forming a recurrent network. The visible units provide the bottom-up input to these V1 complex cells, and they encode disparity signals which in the brain are computed by combining monocular left and right eye simple cells based on phase-shift or position-shift mechanisms [36]. The input from visible units, or the corresponding signals in the brain, are assumed to be "decorrelated" across space when stimulus correlation is factored out [28]. The prior of natural scenes is assumed to be captured by the lateral connectivity among hidden units in the model or among disparity-tuned V1 neurons in the brain. These intrinsic horizontal connections can give rise to noise correlation and other correlated activities among neurons [21, 76, 153].

2.3.1 First order properties of learned hidden units

Figure 2.4 shows typical tuning curves of the hidden units obtained from the model simulation of neurophysiological experiments (Section 2.2.2.2), and the distribution of bias terms $\vec{\alpha}, \vec{\gamma}$. Hidden units shared the same preferred disparity and the general shape as their corresponding input visible units. The bias terms are negative, indicating that the hidden units tend to fire sparsely.

2.3.2 Comparison with computational models in terms of connectivity constraints

The learned lateral connections $\vec{\beta}$ among hidden units form what we call the **disparity association field**, analogous to the well-known contour association field for orientation-tuned neurons [33]. The lateral connectivity, or the disparity association field, observed in the trained Boltzmann machine model is qualitatively in agreement with the cooperative and competitive circuits predicted by Marr and Poggio [110], and with the recent model of Samonds et al. [147] which has been successful in more accurately accounting for neurophysiological data of disparity-tuned neurons in V1.

We define the disparity association field of a hidden unit as the set of lateral connections between it and other hidden units. Figure 2.5a illustrates the disparity association field of one unit tuned near zero disparity in the center column of the 5×5 grid, showing its lateral connections $\vec{\beta}$ to all other units in the network along a particular direction in the grid. The x-axis indicates different hypercolumns or spatial locations, and the y-axis indicates units with different disparity tunings.

The disparity association field learned by the Boltzmann machine has a number of noteworthy features. First, in terms of **inter-columnar connections**, i.e. connections between a unit with units in other hypercolumns, units with the same or similar disparity tunings tended to form positive connections across hypercolumns (spatial receptive field locations) and units with very

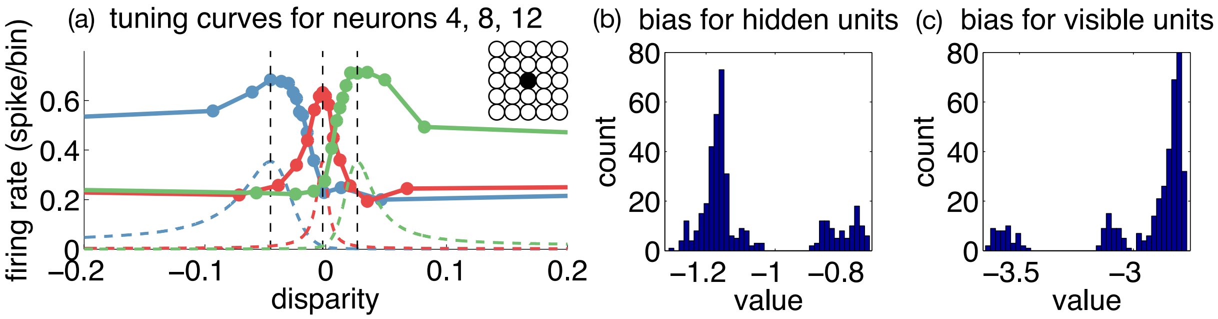


Figure 2.4: Some first order properties of the trained Boltzmann machine model. **(a)** tuning curves of hidden units 4, 8, 12 in column 1 of the model (inset on top right), shown as solid curves, with corresponding tuning curves of their input units shown as dashed curves. Tuning curves for hidden units were computed using the mean firing rates during simulation at different testing stimuli (dots on curves). **(b)** Histogram of bias terms for hidden units. **(c)** Histogram of bias terms for visible units. The cluster of values around -0.8 for hidden units and that around -3.6 for visible units mostly came from units tuned to two ends of the disparity distribution (large negative / positive disparities), which was due to a border effect of the theory in Ganguli and Simoncelli [41].

different disparity tunings formed negative connections. Figures 2.5b and 2.5c show in greater detail how each unit in one hypercolumn was connected to units of various disparity tunings in other hypercolumns. The dark bold line highlights that unit 8 in one hypercolumn formed positive (excitatory) connections to similarly tuned units (units 6, 7, 8, 9) in the other hypercolumns, and negative (inhibitory) connections to units tuned to very different disparities. Second, in terms of **intra-columnar connections**, i.e. connections between units in the same hypercolumn, units exhibited excitation for very similarly tuned units in the same hypercolumn, but exerted a suppressive effect on units of dissimilar tuning properties, as shown in Figure 2.5d. These properties of inter- and intra-columnar connections are roughly consistent with the cooperation between neurons of similar disparities across space (the so-called continuity constraint), and the competition among neurons of different disparities at the same spatial location (the so-called uniqueness constraint) in Marr and Poggio [110]'s classical stereopsis model for solving the correspondence problem.

However, the lateral connectivity exhibited by the Boltzmann machine model was richer than that in Marr and Poggio [110]'s model. First, in terms of intra-columnar connections, in addition to the negative (competitive) intra-columnar connections in Marr and Poggio [110]'s model (Figure 2.7a, blue), our model also learned positive intra-columnar connections among units of similar tunings (Figure 2.7b). In this aspect, our model is more consistent with the model in Samonds et al. [147], which assumes that the intra-columnar interaction has a center excitatory (cooperation between similar neurons) surround inhibitory (competition between dissimilar neurons) profile. This profile is more biologically realistic than that of Marr and Poggio [110], taking into account the overlapping nature of tuning curves within a hypercolumn, and the model in Samonds et al. [147] has been shown to explain well the temporal evolution of a number of tuning properties of V1 disparity-tuned neurons.

Second, in terms of inter-columnar connections, Marr and Poggio [110]'s model only specifies positive inter-columnar connections between neurons of the same tuning (Figure 2.7a, red), implicitly making the strong assumption that the surfaces of the world are all fronto-parallel. However, surfaces in natural scenes are more diverse, characterized with a variety of surfaces such as slants and tilts, convexities and concavities. This richness in natural scene surface structures likely induced the greater variety of inter-columnar connectivity observed in our model (Figure 2.7c) that captures the 3D surface priors to a higher degree than connectivity constraints made in the works of Marr and Poggio [110] and Samonds et al. [147]. Our model is likely more consistent with more advanced computational models for stereopsis that take into account slant, tilt, and curvature [6, 101, 131].

The learned disparity association fields obviously depend on training natural scene data. Figure 2.6 shows the association field obtained by training with data from forest scenes in the Brown data set, with all other parameters unchanged. While the association field along the horizontal direction was symmetrical (Figure 2.6a), the one along the the vertical direction (Figure 2.6b) was skewed, which was not the case for the model trained with interior scenes discussed above (data not shown). This was because the lower parts of forest images are nearer to the viewer than the upper parts, due to the receding ground plane toward the horizon in the forest depth images. This asymmetry along different directions is an interesting prediction that can be tested experimentally.

2.3.3 Comparison with neurophysiological data in terms of functional connectivity

We also compared our model with neurophysiological data in terms of functional connectivities derived from CCH measures (Sections 2.2.2.2 and 2.2.3.2), and they match qualitatively in three aspects.

First, neurophysiological data in our earlier studies [145] suggested that functional connectivity between a pair of disparity-tuned neurons varied as a function of tuning similarity and the distance between the neuronal pair. Functional connectivity is often measured in terms of the peak of the cross-correlogram (CCH peak) and alternatively the area under the CCH peak within a certain (e.g. $\pm 10\,\mathrm{ms}$) window (CCH measure). CCH peak tends to reflect mono-synaptic connections, and CCH measure tends to reflect effective connectivity between a pair of neurons via many possible direct monosynaptic and indirect polysynaptic horizontal or even recurrent feedback connections. Samonds et al. [145] found that CCH peak and CCH measure were both positively correlated with tuning similarity (measured as the Pearson correlation between the tuning curves), with CCH measure data shown in Figure 2.8b. On the other hand, the CCH peak was found to drop with the distance between the centers of the receptive fields, i.e. negatively correlated with RF distance within a visual angle of $1.5°$. However, the CCH measure remained relatively constant, i.e. uncorrelated with RF distance within $1.5°$ (Figure 2.8d). This suggested that while the monosynaptic connections between neurons might be fairly local, the effective connections between disparity-tuned neurons are relatively extensive. Last, while functional connectivities existed between neurons of a variety of tuning similarities at close proximity, significant CCH measures could be observed mostly between similarly tuned neurons with RF distance $> 1°$, shown in Figure 2.8f.

For comparison, Figures 2.8a,c,e show the corresponding results from the Boltzmann machine model. Since there was no synaptic delay in our simulation method, the correlation measures (labeled as CCH measure in Figures 2.8a,c,e for consistency) we obtained from the model were more comparable to the CCH measure, rather than the temporally precise CCH peak in the physiological data. Indeed, we found that the CCH measure positively correlated with tuning similarity (Figure 2.8a) but did not change with RF distance within $1.5°$ though it did drop gradually over a larger distance (Figure 2.8c, pairs with positive CCH measures). This is consistent with the lack of drop in the CCH measure with RF distance in the neural data within $1.5°$ as shown in Figure 2.8d. The model predicted a drop in the CCH measure over a larger RF distance but currently data on long-range CCH measures between disparity-tuned neurons are not available. However, similar studies by Smith and Kohn [153] on orientation-tuned neurons did show that the CCH measure dropped only beyond $2\,\mathrm{mm}$ in cortical distance between neurons while the CCH peak dropped even at $0.5\,\mathrm{mm}$. Thus, we anticipate that the CCH measure between disparity-tuned neurons will drop beyond $1.5°$ as our model predicted, but this remains to be tested experimentally. For RF distance vs. tuning similarity (Figure 2.8e), units of similar tuning properties exhibited stronger positive functional connectivities than units of dissimilar tunings when the distance between units became larger.

While all these are in general agreement with the neurophysiological data, there were some differences. Most notably, the model exhibited negative functional connectivities between pairs of units of all distances and similarities (Figure 2.8a,c), whereas the neural data only showed positive connectivities. These differences were largely due to the limitation of the more abstract Boltzmann machine in approximating real neural circuits in both architecture and dynamics as discussed in Section 2.4.2.

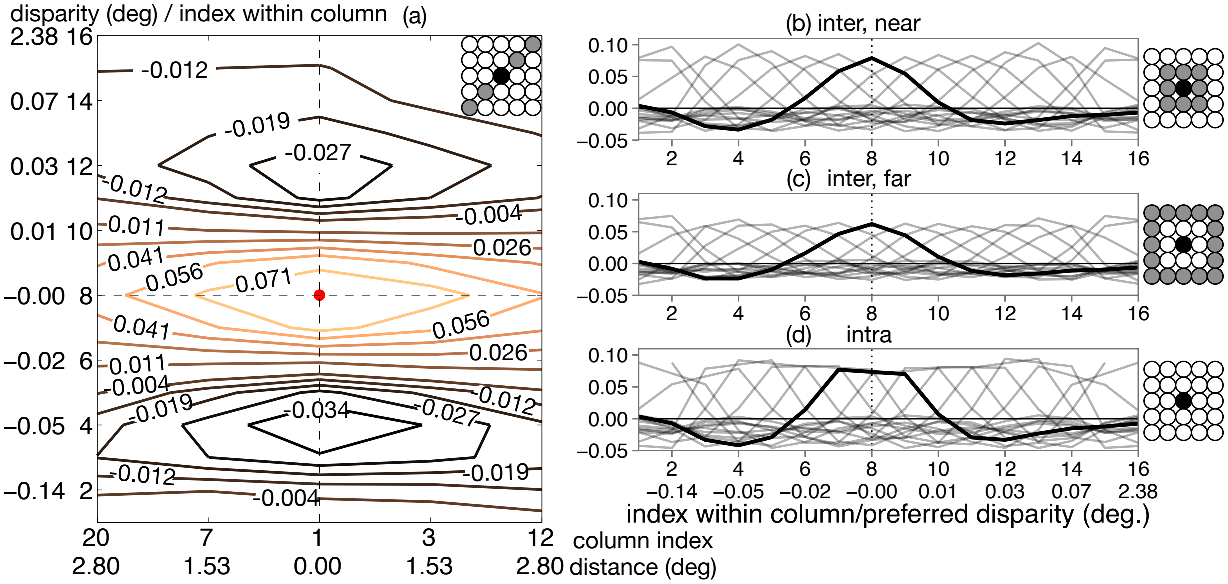


Figure 2.5: Disparity association field. **(a)** disparity association field from unit 8 in the central hypercolumn (column 1) to all other units in 5 hypercolumns (inset on top right; black for column 1, and gray for all 5 but column 1), shown in a contour map of lateral connections. The shapes of contour lines resemble those in the contour association field [33]. The horizontal axis has two rows, first column index, and second distance to column 1; the vertical axis has two columns, from left to right being the preferred disparity of units, and index of the unit in its hypercolumn (1 to 16). **(b)** Inter-columnar connections from column 1 to 8 nearby columns. Each curve shows the average connections between one unit in column 1 to unit of a certain index in other 8 columns (inset on right; black for column 1, and gray for other 8). The curve for unit 8 in the central column is highlighted for clarity. Its value at, say, index $= 9$ is the average of connections from unit 8 in the central column to every unit 9 in surrounding 8 columns, and so on. **(c)** Inter-columnar connections from column 1 to 16 columns further away (inset on right). The connections were generally weaker than those in the panel above, due to longer distances between columns. **(d)** Intra-columnar connections within column 1 (inset on right), with the curve for unit 8 highlighted for clarity. Units of very similar tunings tended to facilitate and help each other, but units of different tunings would inhibit each other.

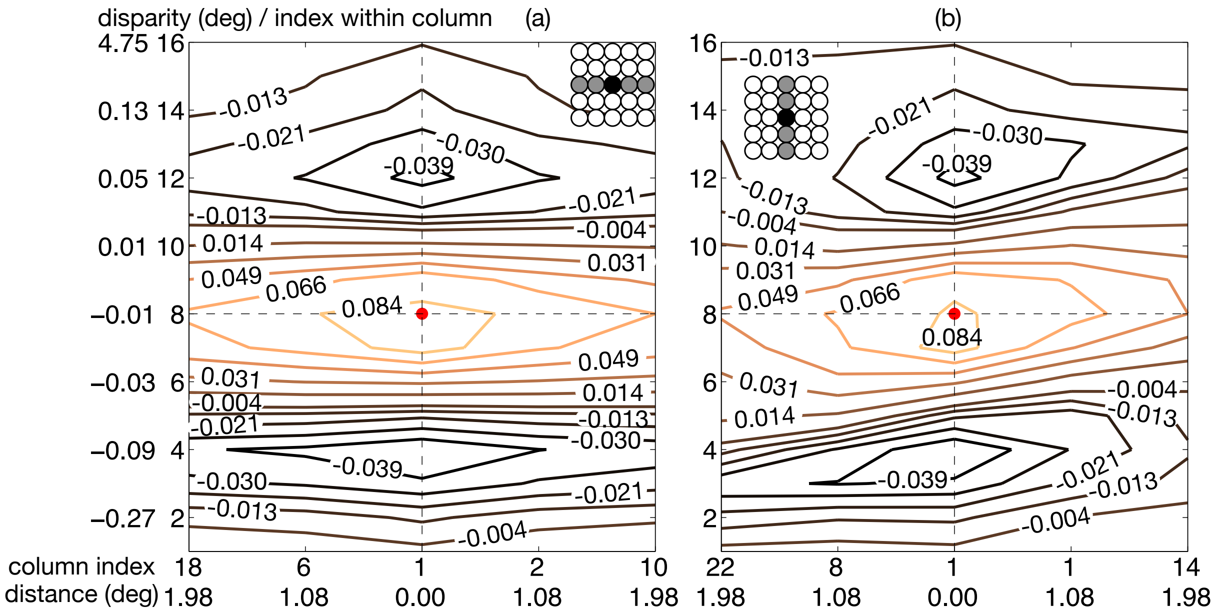


Figure 2.6: Disparity association fields trained using forest images in the Brown data set. **(a)** the association field along 5 hypercolumns horizontally. **(b)** the association field along 5 hypercolumns vetically.

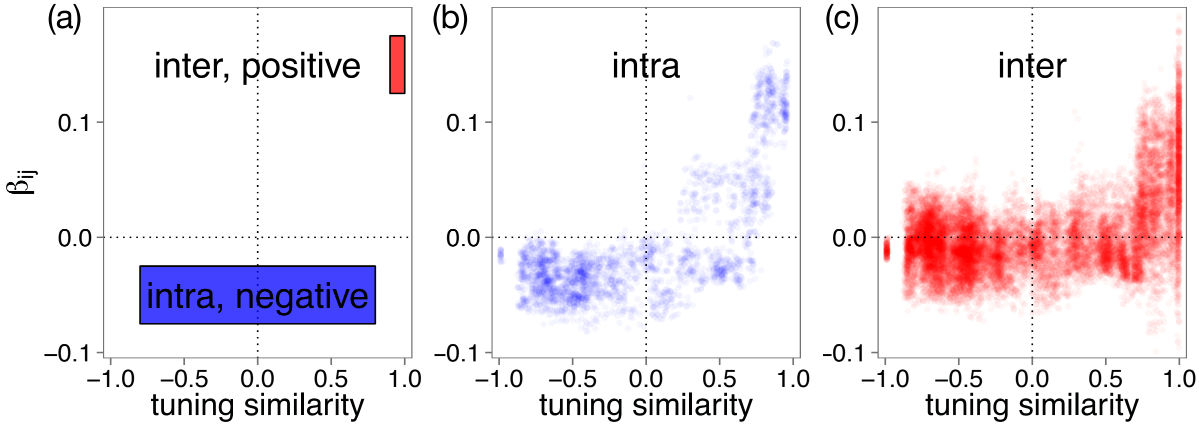


Figure 2.7: Comparison of connectivity in Marr and Poggio [110]'s model and in our Boltzmann machine model, in terms of scatter plot of connection vs. tuning similarity between neurons. **(a)** Schematic of Marr and Poggio [110]'s model, with negative intra-columnar connections between all neurons of different tunings (blue), and positive inter-columnar connections only between neurons of the same tuning (red). **(b)** Intra-columnar connections of our model. **(c)** Inter-columnar connections of our model. The tuning similarity between pairs of hidden units is defined as the Pearson's correlation coefficient between their tuning curves measured from the model simulation (Section 2.2.2.2).

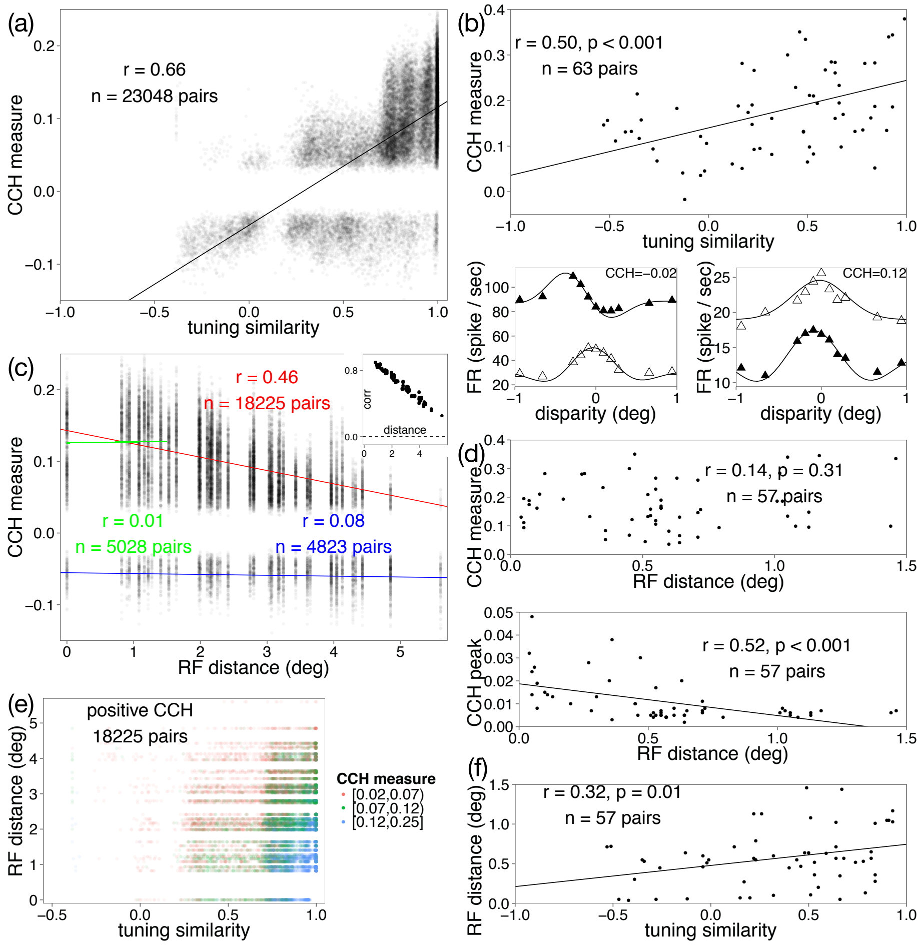


Figure 2.8: Comparison of model and neurophysiological data in terms of CCH measures. **(a)-(b)** CCH measure vs. tuning similarity for model (a) and neural data (b). CCH measure and tuning similarity were positively correlated for both cases. The tuning similarity for a pair of neurons (as well as units) is defined as the Pearson's correlation coefficient of their tuning curves, with two examples shown here in the lower part of (b). The right pair of neurons has a larger similarity than the left one, yielding a larger Pearson's correlation. The more similar pair exhibited a stronger CCH measure (0.12 vs. -0.02). **(c)-(d)** CCH measure vs. distance between neurons for model (c) and neural data (d, upper). In (c), linear fits for pairs with positive (red) and negative (blue) CCH measures are shown separately, and the scatter plot of correlation vs. distance for raw disparity signals is shown in the inset. In addition, a linear fit for positive CCH pairs within $1.5°$ is shown separately (green) to match the range of distance available in the neural data. The CCH measure of the neural data remained constant within this range of RF distance, i.e. uncorrelated with distance, consistent with the model prediction. In (d), CCH peak vs. distance is also shown in the lower panel. **(e)-(f)** distance between neurons vs. tuning similarity for model (e) and neural data (f). For the model, pairs with positive CCH measures were divided into three groups of the same size, and shown in three different colors. Negative CCH pairs were ignored because they were small in magnitude and uniform across different tuning similarities and distances. (b,f) and the lower panel of (d) are adapted from Samonds et al. [145] with permission of authors.

2.4 Discussion

2.4.1 Linking 3D natural scenes and functional connectivity

By training a Boltzmann machine with 3D scene data and simulating neurophysiological experiments on it, this work provides a link between the statistical structure of 3D natural scenes and the measured functional connectivity among disparity-tuned neurons in the primary visual cortex.

We show that certain characteristics of the observed functional connectivity, such as its positive correlation with tuning similarity (Figure 2.8b), its decay over receptive field distance between neurons (Figure 2.8d), and the prevalence of long range excitatory connections between similarly tuned neurons (Figure 2.8f), can be predicted by a Boltzmann machine model trained with natural scene data. The cooperative and competitive connectivity among hidden units in the model is in general agreement with the connectivity constraints in the classical computational model for stereopsis [110] and those in the recent neural circuit model of Samonds et al. [147].

In previous studies on natural scenes, the learning of cooperation between neurons usually relies on co-occurrence [31, 33, 42, 92, 151], with or without additional supervision signals. Hebbian learning can be used to wire up neurons whose encoded patterns co-occur frequently. Competition is often specified manually or by some hypothetical "anti-Hebbian learning rule". Our work demonstrates that Boltzmann machines can provide a coherent computational framework to learn facilitatory connections between disparity-tuned neurons based on co-occurrence statistics of disparity signals, as well as the intra-columnar inhibitory connections that implement the uniqueness constraint in earlier computational models [110], implemented with "hand-crafted" negative connections. Here, we show that if the visual cortex functions mathematically in the manner of a Boltzmann machine or a Markov random field in general, it can acquire these computational constraints—both cooperative and competitive connections—in a unified framework by learning an internal model to explain the input data it experiences during development. The fact that the simulated functional connectivity of the trained Boltzmann machine matches qualitatively with the observed functional connectivity between disparity-tuned neurons (Section 2.3.3) suggests that the interactions among disparity-tuned neurons might form a network, which we call the disparity association field, that encodes the statistics of 3D natural scenes and serves as a prior for solving 3D perception problems. It provides some support to the tantalizing hypothesis that the visual cortex might be functioning like a generative model, e.g. a Boltzmann machine or Markov random field, for statistical inference [98, 99, 100]

There is compelling neurophysiological evidence that V1 disparity-tuned neurons are engaged in horizontal recurrent interactions for disambiguation and surface filling-in [148]. However, the precise biological mechanisms for implementing the Boltzmann machine are not completely clear currently. For the facilitatory connections, Hebbian learning of neurons driven by correlated stimulus signals has been demonstrated in recent in vivo rodent experiments [83, 84]. Inhibitory connections may be learned by long term depression or hemostatic synaptic scaling mechanisms, and the fairly spatially extensive inhibitory interactions in the model may be implemented through global inhibitory neurons (see Section 2.4.2 for more discussion).

As an aside and clarification, we want to point out the differences between the functional connectivity (CCH) in neurophysiology [2, 145] and the lateral connection ($\vec{\beta}$ in Eq. (2.4)) in the Boltzmann machine, or other computational models in general [110, 147]. While they are visually comparable in our study (e.g. Figures 2.8a vs. 2.7c), they are mathematically different in nature: CCH reflects correlation whereas $\vec{\beta}$ reflects partial correlation or inverse covariance, one (approximately) being the inverse of the other [38]. The precise relationship between them requires further investigation.

2.4.2 Limitations of the model

One notable discrepancy between the trained Boltzmann machine model and the neural data mentioned in Section 2.3.3 is that the model had many short-range and long-range negative functional connectivities between hidden units, while the functional connectivity measured between neurons tend to be positive (Figures 2.8a,c vs. Figures 2.8b,d). There are two possible causes for the discrepancy.

First, the input stimuli in the neurophysiological experiments were $8\,\mathrm{Hz}$ dynamic random dot stereograms, and the refresh of the stimulus pattern every $125\,\mathrm{ms}$ drove the neurons synchronously which could induce a bottom-up positive correlation which might cancel out or overshadow the pairwise negative interaction between neurons.

Second, and possibly more significantly, the brain is not likely to implement extensive local and long-range pairwise negative connections between neurons. Inhibition in the cortical circuitry tends to be mediated by local mechanisms, typically within a hypercolumn. Thus long range inhibition is likely mediated by a cooperation between long-range excitatory connections and local inhibitory neurons. There are numerous types of inhibitory neurons in each hypercolumn, mediating a variety of physiological phenomena such as normalization and surround suppression. The Boltzmann machine offers an interesting proposal, among others, on what mathematical model the cortical connections might be implementing. It would be interesting to explore to what extent the neurons in the visual cortex can implement all the short- and long-range excitations and inhibitions suggested by the Boltzmann machine. In the Boltzmann machine, as in any typical neural network, units can exert both excitation and inhibition on one another. This clearly violates the Dale's law under which a neuron can only be excitatory or inhibitory but not both. There are, however, many examples of recent neural circuit models with excitatory and inhibitory pools of neurons that could be extended to implement neural network models with greater biological realism [79]. We are currently investigating an implementation of the model that obeys Dale's law, as well as other realistic biological constraints.

Our Boltzmann machine model was designed to learn only the pairwise connectivity to capture pairwise correlation structures between disparity-tuned neurons across space. It is thus limited in the types of priors that can be encoded. It encodes 3D scene priors in the form of association fields between disparity values across spatial locations, not association fields between 3D surface elements and shapes. Therefore, it cannot, for example, explicitly encode priors on surface structures such as surface slopes (slants and tilts) or surface curvatures. Another layer of hidden units (presumably corresponding to V2 neurons) receiving feedforward connections from disparity-tuned units corresponding to V1 would be required to encode surface priors in the form of the spatial activation patterns in the V1 population. The disparity association field can perform a certain degree of filling-in and surface interpolation as the association field network model showed in Samonds et al. [147]. An association field for completing curved or slant-tilt surfaces could be implemented in V2 as conjectured in Li and Zucker [101] and Zucker [186].

In addition, neurophysiological experiments on our Boltzmann machine model were simulated with Gibbs sampling, which does not strictly follow or exhibit the dynamics of integrate-and-fire neurons in typical neuronal circuit models. However, there have been recent proposals showing that the temporal variability of neuronal spiking activity could potentially be interpreted as Monte Carlo sampling [35, 59]. In addition, there have been rigorous mathematical and computational studies on the implementation of sampling algorithms in integrate-and-fire neuronal circuits [11, 117]. It is intriguing to contemplate to what extent spike neural networks could actually be implementing the mathematics of the Boltzmann machine.

The comparison between the Boltzmann machine and the neural data we made could only be qualitative for several reasons. We assumed the data set matches the natural experience of the monkeys, and the fixation distribution is uniform. We assumed the distribution of tuning curves for visible units can be derived from the approach in Ganguli and Simoncelli [41] and that spike trains can be modeled as independent Poisson processes. There are a number of "hyperparameters" in our model, such as the number of units per hypercolumn, and the mean firing rate of units. There is no guarantee that our assumptions are correct, all of which would affect the model's quantitative predictions.

The neural data used were noisy, subject to measurement errors, sampling errors, and are amenable mostly to qualitative comparison. Therefore, we seek mostly to demonstrate that Boltzmann machines can be used to predict qualitatively the functional circuitry of the disparity-tuned neurons in the primary visual cortex, and that natural scenes can predict a general pattern of cooperative and competitive connectivity that we call the "disparity association field". While the learned disparity association field can vary with the assumptions and hyperparameters used quantitatively, the qualitative result in terms of cooperation between similarly tuned units and competition between dissimilarly tuned ones would still hold.

2.4.3 Summary

The key findings of this study are as follows. First, certain aspects of cortical circuits can be predicted from Boltzmann machines trained on natural scene data. Second, the cortical circuit among disparity-tuned neurons, by virtue of encoding structures in 3D natural scenes, appears to form a disparity association field that could be useful for stereo processing such as removing ambiguity in solving the correspondence problem or performing surface filling-in or interpolation, as some of our recent experiments indicated [148]. Third, the structures of the intra-columnar and inter-columnar inhibitory interactions learned by our model suggest that Boltzmann machines might provide an alternative perspective on some prevalent neurophysiological phenomena such as normalization and surround suppression [14]; additional studies will be required to confirm this conjecture. Finally, this work suggests that Boltzmann machines are a viable model for understanding how the Bayesian prior of natural scenes is encoded in the visual cortex. By demonstrating the potential relevance of Boltzmann machines for understanding neural circuitry, this work suggests that a broader class of computational models, called Markov random fields, which are popular and widely used in computer vision and have enjoyed considerable success in solving real early vision problems, might be a viable model of the visual cortex. This work points to the exciting possibility that insights from computer vision on this class of models can be leveraged to understand what problems the visual cortex could be solving and the computational architecture and algorithms underlying the solutions of these problems.

Chapter 3

Convolutional neural network models of V1 responses to complex patterns

In this study, we evaluated the convolutional neural network (CNN) method for modeling V1 neurons of awake macaque monkeys in response to a large set of complex pattern stimuli. CNN models outperformed all the other baseline models, such as Gabor-based standard models for V1 cells and various variants of generalized linear models. We then systematically dissected different components of the CNN and found two key factors that made CNNs outperform other models: thresholding nonlinearity and convolution. In addition, we fitted our data using a pre-trained deep CNN via transfer learning. The deep CNN's higher layers, which encode more complex patterns, outperformed lower ones, and this result was consistent with our earlier work on the complexity of V1 neural code. Our study systematically evaluates the relative merits of different CNN components in the context of V1 neuron modeling.

3.1 Introduction

There has been great interest in the primary visual cortex (V1) since pioneering studies decades ago [63, 64, 65]. V1 neurons are traditionally classified as simple and complex cells, which are modeled by linear-nonlinear (LN) models [52] and energy models [1], respectively. However, a considerable gap between the standard theory of V1 neurons and reality has been demonstrated repeatedly, at least from two aspects. First, although standard models explain neural responses to simple stimuli such as gratings well, they cannot explain satisfactorily neural responses to more complex stimuli, such as natural images and complex shapes [25, 54, 87, 171]. Second, more sophisticated analysis techniques have revealed richer structures in V1 neurons than those dictated by standard models [15, 143]. As an additional yet novel demonstration of this gap, using large-scale calcium imaging techniques, we [102, 160] have recently discovered that a large percentage of neurons in the superficial layers of V1 of awake macaque monkeys respond strongly to highly specific complex features; this finding suggests that some V1 neurons act as complex pattern detectors rather than Gabor-based edge detectors as dictated by classical studies [26, 70].

While our previous work [160] has shown the existence of complex pattern detector neurons in V1, a quantitative understanding of the relationship between input stimuli and neural responses for those neurons has been lacking. One way to better understand these neurons quantitatively is to build computational models that predict their responses given input stimuli [177]. If we can find a model that accurately predicts neural responses to (testing) stimuli not used during training, a careful analysis of that model should give us insights into the computational mechanisms of the modeled neuron(s). For example, we can directly examine different components of the model [112, 113, 132], find stimuli that maximize the model output [78, 119], and decompose model parameters into simpler, interpretable parts [125, 139].

A large number of methods have been applied to model V1 neural responses, such as ordinary least squares [25, 163], spike-triggered average [163], spike-triggered covariance [143, 164], generalized linear models (GLMs) [75, 128], nested GLMs [112], subunit models [172], and artificial neural networks [132]. Compared to more classical methods, convolutional neural networks (CNNs) have recently been found to be more effective for modeling retinal neurons [78] and V1 neurons in two studies concurrent to ours [13, 113]. In addition, CNNs have been used for explaining inferotemporal cortex and some other areas [89, 178, 179]. Nevertheless, existing studies mostly treat the CNN as a black box without analyzing much the reasons underlying its success relative to other models, and we are trying to fill that knowledge gap explicitly in this study.

To understand the CNN's success better, we first evaluated the performance of CNN models, Gabor-based standard models for simple and complex cells, and various variants of GLMs on modeling V1 neurons of awake macaque monkeys in response to a large set of complex pattern stimuli [160]. We found that CNN models outperformed all the other models, especially for neurons that acted more like complex pattern detectors than Gabor-based edge detectors. We then systematically explored different variants of CNN models in terms of their nonlinear structural components, and found that thresholding nonlinearity and max pooling, especially the former, were important for the CNN's performance. We also found that convolution (spatially shifted filters with shared weights) in the CNN was effective for increasing model performance. Finally, we used a pre-trained deep CNN [152] to model our neurons via transfer learning [13], and found that the deep CNN's higher layers, which encode more complex patterns, outperformed lower ones; the result was consistent with our earlier work [160] on the complexity of V1 neural code. While some of our observations have been stated in alternative forms in the literature, we believe that this is the first study that systematically evaluates the relative merits of different CNN components in the context of V1 neuron modeling.

3.2 Stimuli and neural recordings

3.2.1 Stimuli

Using two-photon calcium imaging techniques, we collected neural population data in response to a large set of complex artificial "pattern" stimuli. The "pattern" stimulus set contains 9500 binary (black and white) images of about $90\,\mathrm{px}$ by $90\,\mathrm{px}$ from five major categories: orientation stimuli (OT; bars and gratings), curvature stimuli (CV; curves, solid disks, and concentric rings), corner stimuli (CN; line or solid corners), cross stimuli (CX; lines crossing one another), and composition stimuli (CO; patterns created by combining multiple elements from the first four categories). The last four categories are also collectively called non-orientation stimuli (nonOT). See Figure 3.1 for some example stimuli. In this study, the central $40\,\mathrm{px}$ by $40\,\mathrm{px}$ parts of the stimuli were used as model input as 40 pixels translated to 1.33 degrees in visual angle for our experiments and all recorded neurons had classical receptive fields of diameters well below one degree in visual angle around the stimulus center [160]. The cropped stimuli were further downsampled to $20\,\mathrm{px}$ by $20\,\mathrm{px}$ for computational efficiency. Later, we use $\boldsymbol{x}_t$ to represent the t-th stimulus as a 20 by 20 matrix, with 0 for background and 1 for foreground (there can be intermediate values due to downsampling), and $\vec{\boldsymbol{x}}_t$ to denote the vectorized version of $\boldsymbol{x}_t$ as a 400-dimensional vector.

3.2.1.0.1 Stimulus type Previous work modeling V1 neurons mostly used natural images or natural movies [13, 25, 78], while we used artificial pattern images [160]. While neural responses to natural stimuli arguably reflect neurons' true nature better, it has the following problems in our current study: 1) public data sets [20] of V1 neurons typically have much fewer images and neurons than our data set, and limited data may introduce bias on the results; 2) artificially generated images can be easily classified and parameterized, and this convenience allows us to classify neurons and compare models over different neuron classes separately (Section 3.2.2.0.1). While white noise stimuli [113, 143] are another option, we empirically found that white noise stimuli (when limited) would not be feasible for finding the correct model parameters (assuming CNN models are correct); see Supplementary Materials.

3.2.2 Neural recordings

The neural data were collected from V1 superficial layers 2 and 3 of two macaque monkeys A and B. For monkey A, responses of 1142 neurons in response to all 9500 (1600 OT and 7900 nonOT) stimuli were collected. For monkey B, responses of 979 neurons in response to a subset of 4605 (800 OT and 3805 nonOT) stimuli were collected due to time constraints. Each stimulus was presented for 5 repetitions for both monkeys. During each repetition, all recorded neurons' responses in terms of $\Delta F/F$ were collected. Later, we use $r_{t,i}^n$ to denote the neural response of the n-th neuron for the t-th stimulus in the i-th trial ($i = 1, \ldots, 5$), r_t^n to denote the average neural response over trials, and $\vec{\boldsymbol{r}}^n$ to denote all the average neural responses for this neuron as a vector. Specifically, we have $n = 1, \ldots, 1142$, $t = 1, \ldots, 9500$ for monkey A and $n = 1, \ldots, 979$, $t = 1, \ldots, 4605$ for monkey B.

3.2.2.0.1 Cell classification The recorded neurons in the neural data had mixed tuning properties [160]: some acted more like complex pattern detectors, some acted more like simple oriented edge detectors, and some had weak responses to all the presented stimuli. To allow cleaner and more interpretable model comparisons, we evaluated model performance for different types of neurons separately (Section 3.5). For example, when comparing a CNN model and a GLM, we computed their performance metrics over neurons that were like complex pattern detectors and those more like simple edge detectors separately, as it is possible that neurons of different types are modeled best by different model classes. To make such per-neuron-type comparison possible, a classification of neurons is required. Here we use the neuron classification scheme in Tang et al. [160]. First, neurons whose maximum mean responses were not above 0.5 ($\max r_t^n \leq 0.5$) were discarded as their responses were too weak and might be unreliable; then, among all the remaining neurons that passed the reliability test, neurons whose maximum mean responses over nonOT stimuli were more than twice of those over OT stimuli ($\frac{\max r_{t_1}^n}{\max r_{t_2}^n} > 2$, where t_1 and t_2 go over all nonOT and OT stimuli respectively) were classified as HO (higher-order) neurons and the others were classified (conservatively) as OT neurons; finally, all the HO and OT neurons were further classified into subtypes, such as curvature neurons and corner neurons, based on ratio tests similar to the one above—for example, an HO neuron was additionally considered as a curvature neuron if its maximum response over curvature stimuli was more than twice of that over non-curvature stimuli. Overall, ignoring the unreliable ones, at the top level, there were OT neurons and HO neurons; OT neurons were further classified as classical and end-stopping (neurons that responded well to short bars but poorly to long ones) neurons; HO neurons were further classified as curvature, corner, cross, composition, and mixed (neurons that failed ratio tests for all the four types of nonOT stimuli) neurons. Figure 3.7 shows example neurons of different classes.

3.2.2.0.2 Recording technique While most other studies use spiking data collected using multi-electrode array (MEA) technologies, we use calcium imaging data [102, 160]. Although MEA-based spiking data are in theory more accurate, calcium imaging techniques can record many more neurons and do not suffer from spike sorting errors. In addition, Li et al. [102] have shown that the calcium imaging technique we used exhibits linear behavior with MEA technologies across a wide range of spiking activities.

3.3 Methods

Here, we describe three classes of models for modeling V1 neurons in our data set. All the models explored in this study can be considered variants of one-hidden-layer neural networks with different constraints and components. By considering them in the framework of one-hidden-layer neural networks (Section 3.3.4), we can easily identify key components that make CNNs perform better. In addition, all the methods here model each neuron separately (no parameter sharing among models fitted to different neurons) and the numbers of parameters of different models are kept to be roughly the same if possible; the parameter separation and equality in model size ensure a fairer comparison among models. For each neuron n from some monkey, all our models take image $\boldsymbol{x}_t$ of size 20 by 20 as input and try to predict the neuron's mean response r_t^n of image t as output. See Section 3.2 for an explanation of the notation.

3.3.1 CNN models

A CNN model passes the input image through a series of linear-nonlinear (LN) operations—each of which consists of convolution, ReLU nonlinearity [91], and (optionally) max pooling. Finally, outputs of the final LN operation are linearly combined as the predicted response of the

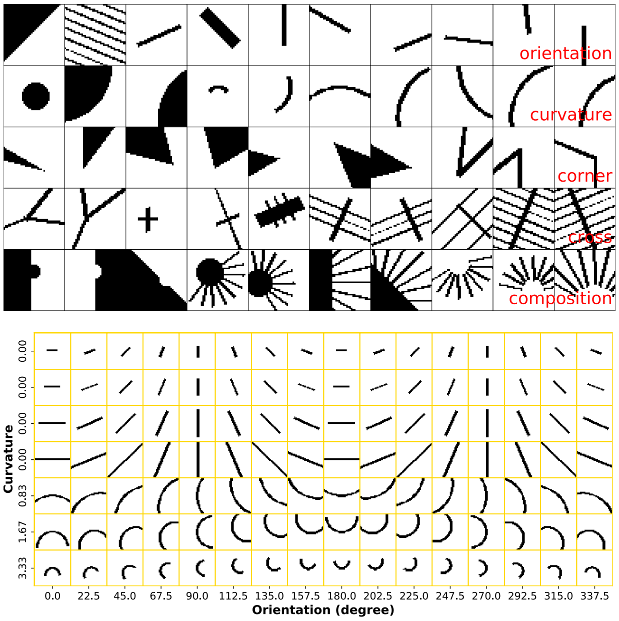


Figure 3.1: **Top** "Pattern" stimulus set. Stimuli are arranged in rows, each row showing 10 randomly drawn stimuli for each of the five categories (see the bottom right corner of each row). Only the central $40\,\mathrm{px}$ by $40\,\mathrm{px}$ parts of stimuli are shown. Refer to Tang et al. [160] for details. **Bottom** A subset of curvature and line stimuli in the stimulus set, ordered by stimulus parameters (curvature, length, and orientation). Only the central $40\,\mathrm{px}$ by $40\,\mathrm{px}$ parts are shown.

neuron being modeled. Our baseline CNN model for V1 neurons is shown in Figure 3.2, with one (convolutional) layer and 9 filters. Given a 20 by 20 input, it first convolves and rectifies ("convolve + threshold" in the figure) the input with 9 filters of size 9, yielding 9 feature maps (channels) of size 12 by 12, one for each filter. Then max pooling operation ("max pool" in the figure) is performed for each feature map separately to produce 9 pooled feature maps of size 4 by 4. Finally, all the individual output units across all the pooled feature maps are linearly combined ("linear combination" in the figure), plus some bias, to generate the predicted neural response.

As shown in Table 3.2 of Section 3.4.1, apart from the baseline model with 9 channels (`B.9` in the table), we also explored other CNN models with the same overall architecture but different numbers of channels.

3.3.2 "Standard" Gabor-based models

Gabor filters are widely used in theoretical models of V1 neurons [24, 26, 70]. Therefore, we tried to fit (relatively speaking) standard Gabor-based V1 models to our data as control. We tried Gabor simple cell models, Gabor complex cell models, as well as their linear combinations (Figure 3.3). Interestingly, to the best of our knowledge, such models were not examined in the existing V1 data fitting literature in terms of their performance compared to more popular ones such as GLMs.

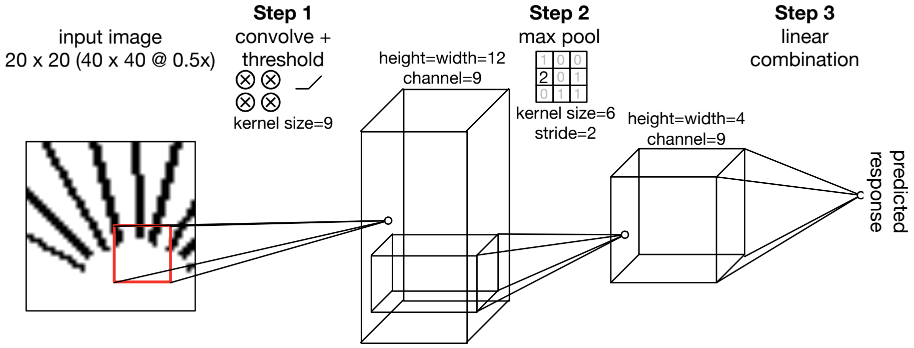


Figure 3.2: The architecture of our baseline CNN model (or `B.9` in Table 3.2). Given a 20 by 20 input image (40 by 40 downsampled to half; see Section 3.2), the model computes the predicted neural response in three steps. In **Step 1** (“convolve + threshold”), the model convolves and rectifies the input to generate an intermediate output of size 12 by 12 by 9 (height, width, channel; 3D block in the middle); concretely, for each of the model’s 9 filters of size 9 by 9 (kernel size), the model computes the dot product (with some bias) between the filter and every 9 by 9 subregion in the input (red square being one example), rectifies ($x \mapsto \max(0, x)$) all the dot products, and arranges the rectified results as a 12 by 12 feature map; the process is repeated for each of the 9 filters (channels) and all the 9 feature maps are stacked to generate the 12 by 12 by 9 intermediate output. In **Step 2** (“max pool”), max pooling operation is performed for each feature map separately to produce 9 pooled feature maps of size 4 by 4; concretely, for each of the 12 feature maps obtained in Step 1, maximum values over 6 by 6 subregions are computed every 2 data points (stride) and arranged as a 4 by 4 pooled feature map; the process is repeated for each of the 9 feature maps to generate the 4 by 4 by 9 pooled output. In **Step 3** (“linear combination”), all the individual output units across all the pooled feature maps are linearly combined plus some bias to generate the predicted neural response. See Section 3.3.1 as well.

3.3.2.1 Gabor simple cell models

A Gabor simple cell model [15, 52, 143] takes the following form:

$$\hat{r} = a[\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{g}}(x, y, \omega, \theta, \sigma_x, \sigma_y, \phi) \rangle + c]_+^2 + b, \tag{3.1}$$

where $\vec{\boldsymbol{x}}$ is the input stimulus in raw pixel space reshaped as a vector (Section 3.2.1), $\vec{\boldsymbol{g}}(x, y, \omega, \theta, \sigma_x, \sigma_y, \phi)$ is the Gabor filter (reshaped as a vector) in the raw pixel space with locations x, y, frequency ω, orientation θ, phase ϕ, and standard deviations of the Gaussian envelope σ_x, σ_y, and a, b, c are scale and bias parameters. In such formulation, given some input, the model computes the dot product between the input and its Gabor filter (plus some bias), and passes the output through a half-wave squaring nonlinearity $[\cdot]_+^2$. In the existing literature, some simple cell models use threshold nonlinearity (also called over-rectification) $[\cdot + c]_+$ while some others use half-wave squaring nonlinearity $[\cdot]_+^2$; according to Heeger [52], these two nonlinearities are approximately the same in some sense and therefore we include them both in our models for more flexibility.

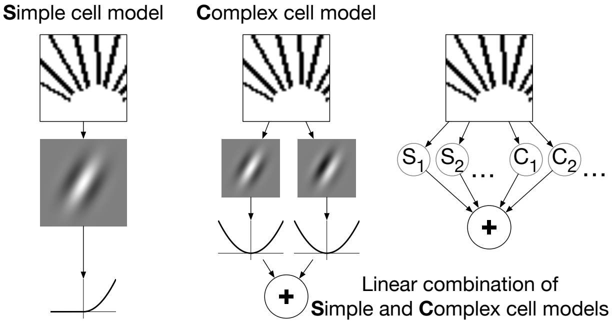


Figure 3.3: The architecture of Gabor-based models. A simple cell model (left) takes the dot product of a Gabor filter and the input, and passes through the output through a half-wave squaring nonlinearity. A complex cell model (middle) takes the dot products of two Gabor filters with quadrature phase relationship, squares and sums the outputs. A linear combination of simple and complex cell models (right) takes some linear combination of some simple cell models (S) and some complex cell models (C). This figure is partially inspired by Figure 1 in Rust et al. [143] and Figure 1 in Carandini et al. [15].

3.3.2.2 Gabor complex cell models

A Gabor complex cell model [1, 15, 52, 143] takes the following form:

$$\begin{aligned}\hat{r} = a[&\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{g}}(x, y, \omega, \theta, \sigma_x, \sigma_y, 0)\rangle^2 \\ &+ \langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{g}}(x, y, \omega, \theta, \sigma_x, \sigma_y, \pi/2)\rangle^2] + b,\end{aligned} \tag{3.2}$$

where $\vec{\boldsymbol{x}}$ is defined as before, $\vec{\boldsymbol{g}}(x, y, \omega, \theta, \sigma_x, \sigma_y, 0)$ and $\vec{\boldsymbol{g}}(x, y, \omega, \theta, \sigma_x, \sigma_y, \pi/2)$ are a pair of Gabor filters (reshaped as vectors) in the raw pixel space with the same parameters (check Section 3.3.2.1 for details) except for phases differing by $\pi/2$, and a, b are scale and bias parameters. In such formulation, given some input, the model computes the outputs of two linear Gabor filters with quadrature phase relationship and sums their squares together to achieve phase invariance. As an aside, while we set phases of the Gabor filter pair to be ϕ and $\phi + \pi/2$ with $\phi = 0$, any other ϕ will also work; empirically we found Eq. (3.2) has no or little dependence on ϕ.

3.3.2.3 Linear combinations of complex and simple cell models

While simple and complex cell models are the canonical ones in most neuroscience textbooks, detailed analyses on monkey V1 neurons have revealed more than one (simple) or two (complex) linear components [15, 143]. Rust et al. [143] call such extensions to “standard” simple and complex cell models “generalized LNP response models” (Figure 1C of Rust et al. [143]). One simple realization of generalized LNP response models is to take linear combinations of “standard” Gabor-based models that are defined above.

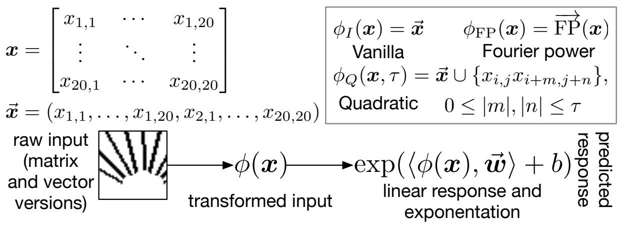

$\boldsymbol{x} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,20} \\ \vdots & \ddots & \vdots \\ x_{20,1} & \cdots & x_{20,20} \end{bmatrix}$
$\vec{\boldsymbol{x}} = (x_{1,1}, \ldots, x_{1,20}, x_{2,1}, \ldots, x_{20,20})$
$\phi_I(\boldsymbol{x}) = \vec{\boldsymbol{x}}$
Vanilla
$\phi_{\mathrm{FP}}(\boldsymbol{x}) = \overrightarrow{\mathrm{FP}}(\boldsymbol{x})$
Fourier power
$\phi_Q(\boldsymbol{x}, \tau) = \vec{\boldsymbol{x}} \cup \{x_{i,j}x_{i+m,j+n}\}$,
Quadratic
$0 \leq |m|, |n| \leq \tau$
raw input (matrix and vector versions)
$\phi(\boldsymbol{x})$
transformed input
$\exp(\langle \phi(\boldsymbol{x}), \vec{\boldsymbol{w}} \rangle + b)$
linear response and exponentation
predicted response

Figure 3.4: The architecture of generalized linear models. The raw input stimulus $\boldsymbol{x}$ is first transformed into $\phi(\boldsymbol{x})$, where different $\phi(\cdot)$ are used for different GLM variants (inside the box). For vanilla GLMs, we use the identity transformation $\phi_I(\cdot)$; for Fourier power models, we use the Fourier power transformation $\phi_{\mathrm{FP}}(\cdot)$ (Section 3.3.3.2); for generalized quadratic models, we use the localized quadratic transformation $\phi_Q(\cdot, \tau)$ (Section 3.4.3.2 and Section 3.3.3.3). The transformed input $\phi(\boldsymbol{x})$ is passed into a linear function $\langle \cdot, \vec{\boldsymbol{w}} \rangle + b$ and the output is exponentiated to give the predicted neural response. For details on the localized quadratic transformation ($\phi_Q(\cdot, \tau)$ in the figure), see Section 3.4.3.2 and Figure 3.5.

3.3.3 Generalized linear models

We consider the following set of Poisson generalized linear models [111, 123] with possibly nonlinear input transformations (Figure 3.4). We also tried Gaussian GLMs and they performed consistently worse than Poisson ones in our experiments. Note that the term "GLM" has been used pretty loosely in the literature, and many models with similar structural components to those in the CNN are considered GLMs by many. We want to emphasize that the purpose of including GLMs in this study is not to compare CNNs and (all the variations of) GLMs in terms of performance but to find key components that make CNN models outperform commonly used models for V1 modeling. We call these models GLMs mainly because they are often formulated as GLMs in the literature. See Section 3.3.4 for the connection between CNNs and GLMs considered in this study.

3.3.3.1 Vanilla generalized linear models

A vanilla (linear) GLM takes the following form:

$$\hat{r} = \exp(\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{w}} \rangle + b), \tag{3.3}$$

where $\vec{\boldsymbol{x}}$ is the input stimulus in raw pixel space reshaped as a vector, $\vec{\boldsymbol{w}}$ is the linear spatial filter in the raw pixel space, and b is the bias parameter for adjusting the firing threshold. The formulation of this vanilla GLM is standard for modeling V1 simple cells [15, 25, 69], which respond to stimuli having appropriate orientation, spatial frequency, and spatial phase [63].

3.3.3.2 Fourier power models

A Fourier power model [25] takes the following form:

$$\hat{r} = \exp(\langle \overrightarrow{\mathrm{FP}}(\boldsymbol{x}), \vec{\boldsymbol{w}} \rangle + b), \tag{3.4a}$$

$$\mathrm{FP}(\boldsymbol{x})(\omega_x, \omega_y) = |\boldsymbol{X}(\omega_x, \omega_y)|^2, \tag{3.4b}$$

where $\mathrm{FP}(\boldsymbol{x})$ computes the Fourier power spectrum of the input stimulus ($\mathrm{FP}(\boldsymbol{x})$ is 2D in Eq. (3.4b) and reshaped to a vector $\overrightarrow{\mathrm{FP}}(\boldsymbol{x})$ in Eq. (3.4a)), $\boldsymbol{X}$ denotes the 2D Fourier transform of the input stimulus, $\vec{\boldsymbol{w}}$ is the linear filter in the Fourier power domain, and b is the bias parameter for adjusting the firing threshold. In practice, Fourier power models provide performance close to the state of the art [15, 87].

3.3.3.3 Generalized quadratic models

A generalized quadratic model (GQM) [124, 125] takes the following form:

$$\hat{r} = \exp(\mathrm{Q}(\vec{\boldsymbol{x}})), \tag{3.5a}$$

$$\mathrm{Q}(\vec{\boldsymbol{x}}) = \vec{\boldsymbol{x}}^T \boldsymbol{W} \vec{\boldsymbol{x}} + \vec{\boldsymbol{a}}^T \vec{\boldsymbol{x}} + b, \tag{3.5b}$$

where $\mathrm{Q}(\vec{\boldsymbol{x}})$ computes a quadratic feature transformation of the (vectorized) input stimulus, $\boldsymbol{W}, \vec{\boldsymbol{a}}, b$ are the second-order parameters, first order parameters, and bias parameter respectively in the transformation. A GQM can be formulated as a GLM with quadratic feature transformation [125], which introduces additional nonlinearity components and flexibility for neuron modeling. In addition, there is a connection between GQMs and spike-triggered based methods under certain conditions [124] and GQMs are statistically more efficient. Note that Fourier power models (Section 3.3.3.2) can be also formulated as GQMs, as $|\boldsymbol{X}(\omega_x, \omega_y)|^2 = (\vec{\boldsymbol{x}}^T U(\omega_x, \omega_y))(\vec{\boldsymbol{x}}^T \overline{U}(\omega_x, \omega_y))$ in Eq. (3.4b) , where $U(\omega_x, \omega_y)$ denotes the Fourier transform vector for frequency pair (ω_x, ω_y) and $\overline{U}(\omega_x, \omega_y)$ denotes its complex conjugate.

3.3.4 Connections among CNNs, Gabor models, and GLMs

As mentioned in the beginning of Section 3.3, the three classes of models considered in this study are connected and form a continuum as they all can be roughly formulated as vanilla one-hidden-layer neural networks [7], or one-hidden-layer multilayer perceptrons (MLPs):

$$\hat{r}(\vec{\boldsymbol{x}}) = \sum_{i=1}^{C} c_i z_i(\vec{\boldsymbol{x}}) + b, \tag{3.6a}$$

$$z_i(\vec{\boldsymbol{x}}) = f(a_i(\vec{\boldsymbol{x}})), \tag{3.6b}$$

$$a_i(\vec{\boldsymbol{x}}) = \langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{w}}^{(i)} \rangle + b_i. \tag{3.6c}$$

A one-hidden-layer neural network computes the output $\hat{r}$ given (vectorized) input stimulus $\vec{\boldsymbol{x}}$ according to Eqs. (3.6). Overall, the output is a linear combination of C hidden units' output values z_i as shown in Eq. (3.6a). Each hidden unit's output is computed by applying some nonlinearity (also called activation function) f on the pre-activation value of the hidden unit a_i as shown in Eq. (3.6b), and pre-activation value a_i is a linear function of input specified by weights $\vec{\boldsymbol{w}}^{(i)}$ and bias b_i as shown in Eq. (3.6c).

Gabor models can be formulated as MLPs with constraints that weights $\vec{\boldsymbol{w}}^{(i)}$ must be Gabor functions. A simple cell model is a MLP with one hidden unit and half-wave squaring nonlinearity; a complex cell model is a MLP with two hidden units in quadrature phase relationship and squaring nonlinearity; a linear combination of simple and complex cell models is a MLP with multiple hidden units and mixed nonlinearities.

GLMs can be formulated as MLPs with an additional exponential nonlinearity on output. A vanilla GLM is a MLP with one hidden unit and no nonlinearity (linear); a Fourier power GLM is a MLP with multiple hidden units of fixed weights (Fourier basis functions) and squaring nonlinearity; A GQM is a MLP with multiple hidden units and squaring nonlinearity—the linear term in Eq. (3.5b) can be absorbed into the quadratic one as long as the quadraic coefficient matrix is full rank. Empirically, we found the additional accelerating exponential nonlinearity to be unimportant for the modeling of our data, as Poisson GLMs with the additional accelerating nonlinearity performed similarly or marginally better, compared to Gaussian and softplus GLMs without such nonlinearity (Supplementary Materials).

A CNN can be formulated as a MLP with ReLU ($x \mapsto \max(0, x)$) nonlinearity and an additional max pooling operation before the final output computation of Eq. (3.6b). Compared to other models, a CNN has additional constraints among the weights of hidden units—shared and spatially shifted in groups. For example, our baseline CNN can be considered as a MLP with $12 \times 12 \times 9 = 1296$ hidden units, as each 9 by 9 filter in the CNN yields a feature map of $12 \times 12 = 144$ hidden units, and there are 9 filters in the CNN. For MLP hidden units derived from a common feature map, filter weights are shared and spatially shifted; for MLP hidden units derived from different feature maps, filter weights are independent. This group-wise sharing of hidden unit weights in CNN models is not present in GLMs, which we will compare in detail with CNNs in Section 3.5 as GLMs were the best-performing non-CNN models in our experiments.

Table 3.1 gives a summary of different models in terms of their structures, under the framework of one-hidden-layer neural network (or MLP). We classify nonlinearities into thresholding (half-wave squaring and ReLU) and non-thresholding (squaring) ones, because we found all the thresholding activation functions behaved essentially the same in our experiments (Section 3.5.2) and we think that being thresholding or not may be the most important aspect for a nonlinearity.

3.3.5 Pre-trained CNNs

There are at least two different ways to model V1 neurons and neural data in general using CNNs: data-driven and transfer learning. In the data-driven approach, CNNs are trained from scratch to fit the neural data. This is the approach taken in this study and many other very recent ones [78, 113]. In the transfer learning (also called goal-driven) approach [89, 179], CNN models are first trained on some other tasks such as image classification, and then neural data are fitted by (linearly) combining outputs of fitted units in the trained models. As shown in Cadena et al. [13], two approaches work similarly for V1 neurons in response to natural images.

As an additional experiment, we tried to model neurons in our data set using a transfer-learning approach similar to that in Cadena et al. [13]. Specifically, we fed all images[1] to the CNN model VGG19 [152] and extracted intermediate feature representations of the images across all the CNN layers (except fully-connected ones). The intermediate representations of

[1] the images were rescaled to $2/3$ of their original sizes; we used this scale because in another study [183] we found that this scale gave the highest representational similarity [90] between the CNN and neural data among all scales explored; we also tried using raw images without rescaling in the current study and got worse results.

Table 3.1: Comparison of model structures for Gabor models, GLMs, and CNNs in the framework of one-hidden-layer MLP. First two columns specify the model class and subclass. The third column shows whether the models' corresponding MLPs have multiple hidden units or not. The fourth column shows the constraints among hidden units imposed by the models; "independent" means weights for different hidden units can vary independently, "shared" means weights for different hidden units are tied together (via convolution), "quadrature phase" means weights of the hidden unit pair are in quadrature phase relationship (specific to Gabor models), and "fixed" means weights are not learned but specified before training. The fifth column specifies the nonlinearity (activation function), with "none" meaning no nonlinearity (identity or linear activation function), and "mixed" meaning both thresholding and non-thresholding nonlinearities. The last column specifies additional structures imposed by the models.

Class	Subclass	Multiple units	constraints among units	nonlinearity	additional structures
	simple	No	—	thresholding	weights are Gabor
Gabor	complex	Yes	quadrature phase	non-thresholding	weights are Gabor
	combination	Yes	independent	mixed	weights are Gabor
	vanilla	No	—	none	exponential output
GLM	Fourier power	Yes	fixed (not learned)	non-thresholding	exponential output
	GQM	Yes	independent	non-thresholding	exponential output
CNN	—	Yes	independent + shared	thresholding	max pooling

each layer were used as inputs to train (a set of) GLMs to model all the neurons. All the other implementation details were the same as those for GLMs (Section 3.4.3).

3.3.6 Model evaluation

Given some model $f_{\vec{\boldsymbol{\theta}}}$ with trainable parameters $\vec{\boldsymbol{\theta}}$, we evaluate its performance $s(f_{\vec{\boldsymbol{\theta}}}, D)$ on a single neuron n based on its input stimuli and responses $D = \{(\vec{\boldsymbol{x}}_t, r_t^n)\}$, where $\{r_t^n\}$ are the across-trial average responses computed from $\{r_{t,i}^n\}$ (Section 3.2.2), by squared normalized correlation coefficient $\mathrm{CC}_{\mathrm{norm}}^2$ [60, 150].

To evaluate a model, we first partition D into training set D_{train} and testing set D_{test} using $80\,\%$ and $20\,\%$ of the whole data set, respectively. For those models involving model selection in the training (all but Gabor models), $20\,\%$ of the training data is reserved for validation purpose. We use D_{train} to obtain the trained model $f_{\vec{\boldsymbol{\theta}}^*}$, and compute the model performance $s(f_{\vec{\boldsymbol{\theta}}}, D)$ as follows (neuron index n is omitted as there is only one neuron being considered):

$$
\begin{aligned}
s(f_{\vec{\boldsymbol{\theta}}}, D) &= \mathrm{CC}^2_{\text{norm}}(f_{\vec{\boldsymbol{\theta}}^*}, D), && (3.7)\\
\mathrm{CC}_{\text{norm}}(f_{\vec{\boldsymbol{\theta}}^*}, D) &= \frac{\mathrm{CC}_{\text{abs}}(\vec{\boldsymbol{r}}_{\text{test}}, \hat{\vec{\boldsymbol{r}}}_{\text{test}})}{\mathrm{CC}_{\text{max}}(\{r_{t,i}\})}, && (3.8)\\
\hat{\vec{\boldsymbol{r}}}_{\text{test}} &= (\ldots, f_{\vec{\boldsymbol{\theta}}^*}(\vec{\boldsymbol{x}}_j), \ldots), && (3.9)\\
\vec{\boldsymbol{r}}_{\text{test}} &= (\ldots, r_j, \ldots) \quad (\vec{\boldsymbol{x}}_j, r_j) \in D_{\text{test}}, && (3.10)\\
\mathrm{CC}_{\text{abs}}(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) &= \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2}\sqrt{\sum_i (y_i - \overline{y})^2}}, && (3.11)\\
\mathrm{CC}_{\text{max}}(\{r_{t,i}\}) &= \sqrt{\frac{\mathrm{Var}(\{\sum_j r_{t,j}\}) - \sum_j \mathrm{Var}(\{r_{t,j}\})}{5(5-1)\mathrm{Var}(\{r_t\})}}. && (3.12)
\end{aligned}
$$

Concretely, we first compute the raw Pearson correlation CC_{abs} between the set of neural responses $\vec{\boldsymbol{r}}_{(k)}$ and the set of model responses $\hat{\vec{\boldsymbol{r}}}_{\text{test}}$ using Eq. (3.11) ($\overline{x}$ and $\overline{y}$ denote mean values of the two inputs), then divide this CC_{abs} by CC_{max}, which is defined in Eq. (3.12) (adapted from Schoppe et al. [150], with 5 in the denominator being the number of trials) and estimates the maximal Pearson correlation coefficient an ideal model can achieve given the noise in the neural data [60, 150] , to get the normalized Pearson correlation coefficient using Eq. (3.8), and finally square $\mathrm{CC}_{\text{norm}}$ to get the model performance $s(f_{\vec{\boldsymbol{\theta}}}, D)$ using Eq. (3.7). As squared CC_{abs} gives the fraction of variance in neural responses explained by the model in a simple linear regression, squared $\mathrm{CC}_{\text{norm}}$ gives the normalized explained variance that accounts for noise in the neural data. Notice that CC_{max} is computed over all the data instead of testing data for more accurate estimation.

Our definition of model performance depends on how D is partitioned. To make our results less susceptible to the randomness of data partitioning, we report results averaged over two partitions.

3.4 Implementation Details

3.4.1 CNN models

3.4.1.1 Detailed model architecture

Table 3.2 shows all the three CNN model architectures we evaluated against other models (Section 3.5), with the baseline CNN model (Figure 3.2) denoted `B.9` in the table. For a fair comparison between CNNs and other models (primarily GLMs; Gabor models inherently have too few parameters), in addition to the baseline CNN model `B.9`, we also evaluated two variants of the baseline model by changing its number of channels. Overall, the three CNN models match the three classes of GLMs (Section 3.4.3) in terms of model size. For vanilla GLMs (401 parameters), we picked the 4-channel CNN architecture (393 parameters); for Fourier power GLMs (around 200 parameters due to the symmetry of Fourier power for real-valued input), we picked the 2-channel one (197 parameters). For GQMs, whose original numbers of parameters are too large, we decided to perform PCA on their input data to reduce the dimensionality. We set the reduced dimensionality to 882 (therefore 883 parameters for GQMs), and evaluated GQMs against the baseline 9-channel CNN architecture (883 parameters). While we could keep more or fewer input dimensions for GQMs and use CNN models with more or fewer channels accordingly, we found that (1) the CNN's performance relatively plateaued for having more than 9 channels (see Supplementary Materials) and (2) keeping fewer input dimensions could potentially affect the performance of GQMs (see Section 3.4.3.2). The 2- and 4-channel CNNs were used mainly for a fair comparison of CNNs and other models, and the baseline 9-channel one was further analyzed.

Table 3.2: CNN model architectures explored in this work. Each row describes one CNN model architecture, with the first column showing its name (`B.n` where `n` is the number of channels), middle columns describing its computational components, and the last showing its number of parameters. Each CNN model first passes the input image through three computational components shown in the table—convolution (conv), nonlinearity, and pooling—and then linearly combine ("fully connected" in CNN jargon) output values of the pooling operation to give the model output. The baseline CNN (`B.9`) has its number of parameters shown in boldface. The number of parameters is computed by adding the number of parameters in the convolutional layer and that in the fully connected layer. For example, the baseline model `B.9` has $9 \times (9 \times 9 + 1) = 738$ parameters (9 for number of channels, 9 for kernel size, and 1 for bias) for the convolutional layer, and $9 \times 4 \times 4 + 1 = 145$ parameters (9 for number of channels, 4 for pooled feature map's size, and 1 for bias) for the fully connected layer, resulting in $738 + 145 = 883$ parameters.

Name	conv	nonlinearity	pooling	# of params
`B.2`	(kernel 9, channel `n`)	ReLU	(max pool, kernel 6, stride 2)	197
`B.4`				393
`B.9`				**883**

3.4.1.2 Optimization

The models were implemented in PyTorch [126], version `0.3.1`. Model parameters were optimized to minimize the mean squared error between model outputs and recorded neural responses. Of all the training data, $80\,\%$ of them were used for actual training, and the remaining $20\,\%$ were kept as validation data for early stopping [46] and model selection. For each combination of neuron and model architecture, we trained the model four times using four sets of optimization hyperparameters (Table 3.3), which were selected from more than 10 configurations in our pilot experiments (Supplementary Materials) conducted on about 10 neurons[2]. Of all the four models trained using different optimization hyperparameters, the one with the highest performance on validation data in terms of Pearson correlation coefficient was selected.

[2] in theory we should exclude these neurons for model evaluation, we did not do it as doing it or not has negligible effects with hundreds of neurons in our data set.

Table 3.3: Optimization hyperparameters for CNN models. Minibatch size was set to 128 in all cases, momentum (for SGD) was set to 0.9, and other hyperparameters, such as β in Adam [80], took default values in PyTorch. LR, learning rate; L2 conv, L2 weight decay on the convolutional layer; L2 fc, L2 weight decay on the fully connected layer; SGD, vanilla stochastic gradient descent with momentum.

Name	Optimizer	LR	L2 conv	L2 fc
`1e-3_1e-3_a002`	Adam	0.002	0.001	0.001
`1e-4_1e-3_a002`	Adam	0.002	0.0001	0.001
`1e-3_1e-3_s1`	SGD	0.1	0.001	0.001
`1e-4_1e-3_s1`	SGD	0.1	0.0001	0.001

3.4.2 "Standard" Gabor-based models

The models were implemented in PyTorch [126]. Input stimuli were preprocessed to have zero mean for each stimulus. Model parameters were optimized to minimize the mean squared error between model outputs and recorded neural responses using Adam [80] without weight decay and with full batch learning (so gradients were computed using the full data set). To (partially) avoid getting trapped in local optima, for each fitted model, we repeated optimization procedures over hundreds of random initializations and took the set of optimized parameters with the smallest error as the final set of optimized parameters. Empirically, such nested optimization procedure converged to ground-truth model parameters almost all the time. Unlike CNNs (Section 3.4.1.2) or GLMs (Section 3.4.3.3), here all the training data were used in the actual training and no model selection was performed, as Gabor models have very few parameters and overfitting should not be a problem.

3.4.2.1 Linear combinations of "standard" Gabor-based models

The implementation details are essentially the same as those in Section 3.4.2. We tried the following combinations of simple cell and complex cell models: one simple plus one complex; one simple plus two complex; two simple plus one complex.

3.4.3 Generalized linear models

3.4.3.1 Vanilla GLMs and Fourier power models

For vanilla GLMs (Section 3.3.3.1), raw stimuli were vectorized into 400-dimensional vectors as model input; for Fourier power models (Section 3.3.3.2), we first applied a Hann window to each raw stimulus as done in David and Gallant [25] to reduce edge artifacts, and then computed 2D Fourier power spectra individually for windowed stimuli as model input.

3.4.3.2 GQMs

GQMs (Section 3.3.3.3) are simply standard GLMs with quadratic transformation on input. A full quadratic transformation over a 400-dimensional raw input vector would result in a vector of more than $80\,000$ dimensions. To make the number of parameters manageable, we performed the following two optimizations for the model input of GQMs.

- Instead of the full quadratic transformation, we performed local quadratic transformations with different "localities" (Figure 3.5). Local quadratic transformations only compute quadratic terms over stimulus pixels that are close enough. For example, a local transformation with locality 2 will only compute quadratic terms over pixels that are at most 2 pixels apart in both horizontal and vertical axes. For this study, we tried localities 2, 4, and 8.
- Even with local quadratic transformations, the input dimensionality is still too high for efficient optimization of model parameters. Therefore, we performed principal component analysis (PCA) on the outputs of local quadratic transformations to reduce their dimensionalities. For a fair comparison of GQM models and our CNN models, 882 dimensions [3] were kept as this would make our GQMs have the same number of parameters as our 9-channel CNN (see Section 3.4.1.1; the 9-channel CNN has 883 parameters, and a GLM with a 882-dimensional input has 883 parameters due to the bias parameter). The dimensionality reduction procedure kept over $95\,\%$ of the variance; if the input dimensionality had been made to align with CNN models with fewer channels, less than $95\,\%$ of the variance would have been kept and the performance of GQMs might have been affected much.

3.4.3.3 Optimization

All the models were implemented using glmnet [39] as Poisson GLMs. Similar to CNNs (Section 3.4.1.2), $80\,\%$ of the training data were used for actual training and the remaining $20\,\%$ were kept as validation data for model selection. L1 regularization was used and the best regularization parameter was selected (out of 100 candidates) by computing model performance on the validation data in terms of Pearson correlation coefficient. For all the three GLM variants, the (transformed) input stimuli have highly correlated dimensions, due to high correlations between adjacent pixels in the original stimuli, and such high correlations in the input made glmnet converge extremely slowly in practice. We worked around this issue by performing full PCA (without reducing the dimensionality) on input stimuli before feeding them to glmnet. Empirically we found this speedup trick made little or no difference to model performance. We also tried Gaussian and softplus GLMs; they performed similarly to or worse than Poisson ones in our experiments (Supplementary Materials).

[3] In practice, we performed PCA only on the pure quadratic terms to reduce their dimensionalities to 432 and concatenated the PCAed 432-dimensional pure quadratic terms with the 400-dimensional linear terms to generate the final 882-dimensional input vectors; such method would guarantee that the information from linear terms, which are heavily used in most V1 models, is preserved. We also tried performing PCA on both linear and pure quadratic terms together and two methods made little difference in our experiments.

$$\phi_Q(\mathbf{x},\tau) = \{x_{i,j}\} \cup \boxed{\{x_{i,j}x_{i+m,j+n}\}}, \quad 0 \le |m|, |n| \le \tau$$

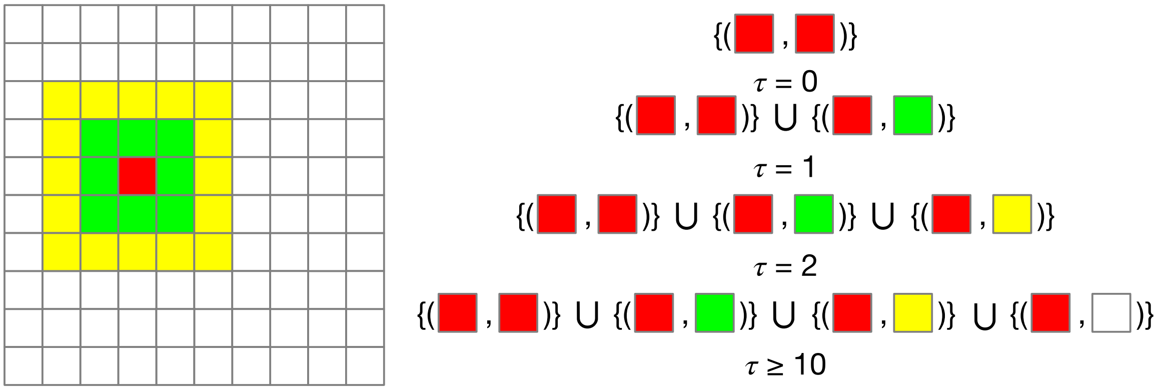


Figure 3.5: Quadratic feature transformation $\phi_Q(\boldsymbol{x},\tau)$ transforms original stimulus $\boldsymbol{x}$, whose elements are indexed by pixel locations i, j, into quadratic features with "locality" τ. The output vector contains the union of first order terms $\{x_{i,j}\}$ and second order terms involving pixels differing by at most τ pixels in all directions, as shown by the equation above. The diagram below shows how to compute the second order terms (shaded box in the equation) of some pixel (denoted in red) in a $10\,\mathrm{px}$ by $10\,\mathrm{px}$ stimulus for different τ's. When $\tau = 0$, only the second order interaction between the red pixel and itself is included; when $\tau = 1$, additional interactions between the red pixel and each green one are included, and so on.

3.5 Results

3.5.1 CNN models outperformed others especially for higher-order neurons

Figure 3.6 shows the performance of CNN models vs. others (except pre-trained CNN models; see Section 3.5.4) on explaining our V1 neural data. Because the full stimulus set consists of different types of stimuli (OT, CN, CV, etc.; see Section 3.2.1), and the full population of neurons for each monkey consists of two subsets (OT neurons and HO neurons, which can be divided into finer subsets as well; see Section 3.2.2) that responded very differently to different types of stimuli, we trained all models using different stimulus subsets ("OT" stimuli and all stimuli; we also tried training only on "nonOT" stimuli, and that gave similar results to using all stimuli), and evaluated each model in terms of its average $\mathrm{CC}^2_{\mathrm{norm}}$ (Section 3.3.6) averaged over OT neurons and HO neurons (for results on finer subsets, see Section 3.5.2 and later). We do not show results of HO neurons trained on OT stimuli, as HO neurons by definition did not respond to OT stimuli well and the results might be unreliable.

We compare CNN models and other models at two different levels. At the individual model architecture level (solid bars in Figure 3.6), we compare specific CNN architectures (models with different numbers of channels) with Gabor models and GLMs. In this case, CNN models with more channels worked better and they outperformed their GLM counterparts (`B.2` vs. Fourier power GLMs, `B.4` vs. linear GLMs, and `B.9` vs. GQMs; see Section 3.4.1.1) across the board;

GQMs had in general better performance than other GLMs, but still fell behind CNNs by a large margin. Gabor models performed similarly to GLMs or worse, and were outperformed by CNNs as well.

At the overall model category level (dashed lines in Figure 3.6), we compare CNN models as a whole to Gabor models as a whole as well as GLMs as a whole. To do this, for each model category, we constructed an "all" model for that category by choosing the best performing model architecture (in terms of performance on validation data for CNNs and GLMs, and in terms of performance on training data for Gabor models; testing data was never used during the model selection) for each individual neuron. By comparing the dashed lines, we have the following empirical observations about the three model classes.

3.5.1.0.1 CNNs outperformed other models especially for HO neurons with complex stimuli When stimuli were the same, the relative performance gap between CNN and other models was larger for HO neurons than OT neurons (middle and right columns of panels of Figure 3.6). For example, on Monkey A, the relative performance increase of the CNN over the GLM increased from $34.2\,\%$ for OT neurons to $52.2\,\%$ for HO neurons. When neurons to model were the same, the relative performance gap was larger for complex stimuli than simple stimuli (left and middle columns of panels of Figure 3.6). For example, on Monkey A, the relative performance increase of the CNN over the Gabor model increased from $27.3\,\%$ for "OT" stimuli to $48.5\,\%$ for all stimuli.

3.5.1.0.2 Priors on Gabor models helped especially with limited data When the stimuli were limited and simple, Gabor models outperformed GLMs, possibly due to the strong and neurophysiologically reasonable prior on Gabor models that filter weights can be described well by Gabor functions [70], and vice versa when the stimuli were relatively sufficient and rich (leftmost column of panels vs. other panels of Figure 3.6). One may hypothesize that multi-component Gabor models (`multi` ones) outperformed standard ones (`complex` and `simple`) mostly due to having multiple orientations; this was not true as shown in Section 3.5.3.

Finally, Figure 3.7 shows the fitting results of some neurons in different classes (see Section 3.2.2); for CNN models, we also show the learned filters and visualization results obtained by activation maximization [119]; these visualization results are images that activate fitted CNNs most. In most cases, Gabor models and GLMs failed to predict the high-responding parts of the tuning curves compared to CNNs.

In Supplementary Materials, we show that CNN models outperformed others even with less amount of data; we also show additional results on CNN models, such as comparison of different optimization configurations and comparison of different architectures (different numbers of layers, different kernel sizes, and so on). We will focus on the one-convolutional-layer CNN model `B.9` with 883 parameters for the rest of this study, because its performance was close to the best among all the CNN models we tried (Supplementary Materials) without having too many parameters, and its one-layer architecture is easier to analyze than those of similarly performing models.

3.5.2 What made CNNs outperform other models

As shown in Figure 3.6, the baseline CNN architecture alone (B.9) outperformed GLMs, which were the best non-CNN models in this study, by a large amount, especially for HO neurons. By comparing the row for the CNN and the rows for GLMs (particular the row for the GQM, as GQMs overall performed better than other GLM variants) in Table 3.1 (Section 3.3.4), we hypothesize that this performance gap was primarily due to the structural components present in the CNN but not in GLMs we studied: thresholding nonlinearity (ReLU), max pooling, and shared weights of hidden units (convolution). To test our hypothesis, we explored different variants of our baseline CNN architecture B.9 in terms of its structural components. The results on thresholding nonlinearity and max pooling are given in this part, and those on convolution are given in the next part. While our GLMs possess an exponentiation nonlinearity which is not present in our CNNs, we found that the exponentiation gave little performance increase than without (Supplementary Materials).

To better understand the utilities of thresholding nonlinearity and max pooling, we explored various variants of the baseline CNN architecture in terms of nonlinearity and pooling scheme. Specifically, we tried all combinations of five different nonlinearities—ReLU (R), ReLU followed by squaring (half-squaring, HS), squaring (S), absolute value (A), linear (no nonlinearity, L)—and two different pooling schemes—max pooling (max), average (mean) pooling (avg)—with other structural components unchanged. Thus, we obtained ten different CNN variants (including the original one) and compared them with the "all" model for GLMs (picking the best model architecture for each neuron), or GLM_all as reference. Results are shown in Figure 3.8 and Figure 3.9, which have the same organization: panels a-c show the performance of all explored models as before, but with $\mathrm{CC}^2_{\mathrm{norm}}$ over OT and HO neurons decomposed into average $\mathrm{CC}^2_{\mathrm{norm}}$ for finer subsets inside OT and HO neurons (Section 3.2.2) to examine model performance in more detail; panels d-f show the neuron-by-neuron comparison of different pairs of models for highlighting. Overall, we have the following observations (letters in the parentheses denote the panels used for highlighting among d-f, if any).

- Thresholding nonlinearities outperformed non-thresholding ones (d,e).
- Thresholding nonlinearities performed similarly (f).
- No consistently better pooling type, but max pooling was more powerful in isolation.
- High correlation between per-neuron and average model performance (almost all panels).

3.5.2.0.1 Thresholding nonlinearities outperformed non-thresholding ones Compared to GLMs we explored in this work, one nonlinear structural component unique to CNNs is ReLU, a thresholding nonlinearity. To understand the usefulness of thresholding nonlinearities in general, we compared four CNN variants with thresholding nonlinearities (R_max, R_avg, HS_max, HS_avg) with four without (A_max, A_avg, S_max, S_avg) and found that thresholding (R, HS) in general helped. This can be seen at two levels. At the level of individual architectures, those with thresholding generally performed better than those without (d, e, and rows 5-8 from the top vs. 1-4 in a-c of Figures 3.8 and 3.9). At the level of model categories, we combined all four thresholding models into one "all" model (T_all) and all four non-thresholding ones as well (NT_all), using the same method as we constructed "all" models in Figure 3.6; we found that thresholding helped as well. Our results suggest that the recorded V1 neurons actually take some thresholded versions of the raw input stimuli as their own inputs. There are at least two ways to implement this input thresholding. First, neurons may have some other upstream neurons as their inputs, each upstream neuron with its own thresholding nonlinearity as modeled in McFarland et al. [112], Vintch et al. [172]. Second, the thresholding may happen at the dendritic tree level, as suggested by Gollisch and Meister [45].

3.5.2.0.2 Thresholding nonlinearities performed similarly While the two thresholding nonlinearities (`R` and `HS`) showed better performance overall, we did not see much difference between the two (f, and `HS_max` vs. `R_max`, `HS_avg` vs. `R_avg` in a-c of Figures 3.8 and 3.9). This observation was consistent with Heeger [52], where the author claimed that these two types of thresholding nonlinearities are both consistent with physiological data and the brain might be using one as an approximation to implement the other.

3.5.2.0.3 No consistently better pooling type, but max pooling was more powerful in isolation While thresholding nonlinearities showed better performance consistently than non-thresholding ones as shown above, the results were mixed for two pooling schemes and depended on nonlinearities, combinations of neurons and stimuli, and monkeys (rows 1-8 from the top, as well as `MAX_all` vs. `AVG_all` that were constructed like `T_all` and `NT_all` above, in a-c of Figures 3.8 and 3.9). We suspect such mixed results were due to the complicated interaction between nonlinearity and pooling. In other words, the contributions of nonlinearity and pooling to model performance do not add linearly. Still, we think max pooling is a powerful computational component per se for modeling neural responses, as max pooling alone without any nonlinearity performed comparably with many other models with pooling and nonlinearity (`L_max` vs. others in a-c of Figures 3.8 and 3.9).

3.5.2.0.4 High correlation between per-neuron and average model performance Figures 3.8 and 3.9 show that different models performed differently. We found that the performance increase/decrease of one model over another one seemed to be *universal*, rather than class- or neuron-specific. We can see this universality from several aspects when two models are compared neuron by neuron (d-f of Figures 3.8 and 3.9). First, there was a high correlation between the performance metrics of individual neurons (high Pearson correlation coefficients r). Second, we performed linear regression on each neuron subclass as well as on all neurons (colored solid lines and black dashed line in the lower right corner of each panel), and found all regression lines were very close.

3.5.3 Convolution was more effective than diverse filters

Apart from thresholding nonlinearity and max pooling explored in Section 3.5.2, CNN models have another unique structural component compared to other models in our study—shared weights among hidden units via convolution—as shown in Table 3.1. In contrast, other models with multiple hidden units (when these models are considered as MLPs; see Section 3.3.4) often have hidden units with independent and diverse weights without sharing ("independent" in

Table 3.1). In this section, we explore the relative merits of these strategies for relating weights of different hidden units—shared weights via convolution vs. independent weights—in terms of model performance, not only for the CNN, but also for other model classes. The results are shown in Figure 3.10, with similar layout to Figures 3.8 and 3.9. We have the following observations (letters in the parentheses denote the panels used for highlighting).

- Multiple diverse filters alone did not help much (d vs. e).
- Convolution helped achieve better performance with the same number of parameters (f).

3.5.3.0.1 Multiple diverse filters alone did not help much To examine the impact of having multiple filters with diverse shapes, we explored two classes of models: Gabor models and CNN models. For Gabor models, we examined three single-filter variants—simple cell model (`Gabor_s`), complex cell model (`Gabor_c`), and the "single-component" Gabor model (`Gabor_single`) constructed from simple and complex cell models similarly to "all" models in Figure 3.6—and one multi-filter variant—one simple two complex (`Gabor_1s2c`; other multi-filter models performed worse as shown in Figure 3.6). For CNN models, we varied the number of channels of the baseline CNN `B.9` from 1 (`B.1`) through 18 (`B.18`).

While the multi-filter Gabor model outperformed both simple and complex cell models by a large margin (a-c,d of Figure 3.10), we found that the single-component model (`Gabor_single`), which takes the better one of simple cell and complex cell models for each neuron, worked almost as well as the multi-filter one (a-c,e of Figure 3.10). While there was still some performance gap between `Gabor_single` and the `Gabor_1s2c`, the gap was relatively small and there was strong correlation between the two models in terms of per-neuron performance (Figure 3.10e). For each neuron, we further compared the learned filters of simple, complex, and multi-filter Gabor models, and found that in some extreme cases, the learned multi-filter Gabor model was degenerate in the sense that it had its simple component dominate its complex components or vice versa (Figure 3.10g; check the caption).

The results for one-channel CNN and the baseline 9-channel CNN are shown in the top two rows of Figure 3.10a-c, and we found that the performance increase (around $20\,\%$ to $50\,\%$) was not proportional to the increase in the number of parameters (around $800\,\%$, or 99 vs. 883 parameters). See Figure 3.6 and Supplementary Materials for more results on the model performance of CNN as we change the number of channels.

3.5.3.0.2 Convolution helped achieve better performance with the same number of parameters As shown in the previous part, having multiple independent filters of diverse shapes was not effective for increasing performance relative to the increase in model size it involved. However, we found that convolution was much more effective, achieving better model performance without increasing the number of parameters. To illustrate this, we compared the baseline CNN's average pooling (`R_avg`) variant, which linearly combines ReLU units, with a multilayer perceptron consisting of one hidden layer of 40 ReLU units (`MLP_40`). To make the two models match in the number of parameters, we performed principal component analysis to reduce the input dimensionality for the MLP to 20; therefore the MLP has $40 \times (20 + 1) + 40 + 1 = 881$ parameters, roughly matching the CNN (883 parameters). The CNN outperformed the MLP by a relatively large margin (a-c,f of Figure 3.10). We also explored the trade-off between input dimensionality and number of hidden units for MLP, with the number of parameters roughly fixed (Figure 3.10h); given roughly the same number of parameters, the CNN, which has convolution, consistently outperformed MLPs of various configurations.

One may argue that the (average) pooling, which was difficult to avoid in our experiments as CNNs would otherwise have too many parameters, helped model performance as well; while such interpretation is possible, it is also helpful to simply consider convolution and pooling collectively as a modeling prior that helps neural response prediction with limited number of parameters and training data. The effectiveness of convolution and pooling could also be due to eye movements during neural data recording; as shown in our previous work [160], the eye movement was in general very small (the standard deviation of the distribution of eye positions during stimulus presentation was in general less than $0.05°$ in visual angle, or $0.75\,\mathrm{px}$ in the $20\,\mathrm{px}$ by $20\,\mathrm{px}$ input space of the CNN) for our data, and such interpretation was less likely.

3.5.4 Data-driven vs. pre-trained CNNs and the complexity of V1 neural code

The results are shown in Figure 3.11. When different CNN layers are compared (Figure 3.11g), overall layer `conv3_1` performed the best (`conv4_1` performed similarly but we prefer layers that are lower and thus easier to analyze). The result was largely consistent with that in Cadena et al. [13]; however, we also observed performance decreases in layers `conv3_2` through `pool3` which were not present in Cadena et al. [13], and we will investigate this in the future. When the best VGG layer and our baseline CNN `B.9` are compared, the VGG layer performed simiarly to the CNN (Figure 3.11a-c), and there was a relatively high correlation between the performance metrics of individual neurons for two models (Figure 3.11d-f). We have also tried other variants of VGG and they performed similarly to or worse than VGG19 (Supplementary Materials).

While our results show that pre-trained CNNs were on par with CNNs trained from stratch, it is possible that pre-trained CNNs would perform much better if they were trained on artificial stimuli as well due to the large difference between the image set used to train VGG19 (natural images) and from our artificial stimulus set. However, such possibility might be not very likely: (1) in our preliminary efforts to apply the state-of-the-art 3-layer CNN architecture in Cadena et al. [13] to model our V1 neurons all together (Supplementary Materials), we found that the 3-layer CNN, within the limit of our hyperparameter tuning, performed similarly to our baseline CNN; (2) Cadena et al. [13] have already established (somewhat) that the 3-layer CNN architecture and pre-trained CNNs perform similarly when all are trained with stimuli of similar nature. On the other hand, we found it interesting that CNNs trained on natural images could be used to effectively predict neural responses on artificial stimuli.

We also visualized units across various layers of the VGG19 network by activation maximization [119] implemented in `keras-vis` [88], and found that these units from layers that matched neural data well (`conv3_1` and `conv4_1`) are tuned to relatively complex image features rather than oriented edges (Figure 3.11h); the visualization results were consistent with our earlier work [160] on the complexity of V1 neural code.

3.6 Discussion

3.6.1 Key components for the success of CNN

In this study, we evaluated a variety of Gabor-based models, generalized linear models, and CNN models for modeling V1 neurons of awake macaque monkeys. These models can be considered as a continuum of regression models in statistics or system identification models in sensory neuroscience [177]; specifically, they can be considered as one-hidden-layer neural networks with different structural components and different degrees of model flexibility (Section 3.3.4). This comparative study allows us to empirically identify some key components that are important for modeling V1 neurons, particularly those neurons with selectivity to higher-order features as identified in Tang et al. [160].

In Section 3.5.2, we evaluated CNN models under different combinations of nonlinearity (ReLU, half-squaring, squaring, and linear) and pooling scheme (max and average), and we found that thresholding nonlinearity and max pooling, which are absent in the best-performing non-CNN models (GLMs) in this study (Table 3.1), were important for the CNN's superior performance relative to other models. In particular, we found that models with thresholding nonlinearities (ReLU and half-squaring) consistently performed better than those without. Interestingly, thresholding nonlinearities such as ReLU and half-squaring are already in classical models of simple cells [3, 52], and pooling (average pooling or max-pooling) of simple cells' responses are also in some models of complex cells [34, 135]. In fact, these models of simple and complex cells were the inspiration to the development of the convolutional neural network architecture [40] in the first place. The Gabor-based models did not perform well mostly because their filters were restricted to Gabor functions whereas filters in GLMs and CNNs could take arbitrary shapes. The CNN provides an elegant way of integrating nonlinearities in models of simple and complex cells with more flexible filters in GLMs; in addition, the CNN allows the linear combination of multiple filters, and such linear combination increases model expressiveness.

When all the models are considered as one-hidden-layer neural networks (Section 3.3.4), there are two strategies for relating weights learned for different hidden units—shared and spatially shifted weights (convolution) and independently learned weights for different units (Table 3.1, "constraints among units"). We evaluated the relative merits of these two strategies in Section 3.5.3 and found that convolution was more effective than having multiple independently learned units both for better performance and fewer model parameters. Our CNN models are similar to subunit models in the literature [64, 112, 143, 172], where V1 neurons take (thresholded) responses of upstream intracortical neurons (with approximately spatially shifted receptive fields) as their inputs and exhibit local spatial invariance, particularly for complex cells. Our CNN models are also consistent with previous V1 modeling work using spike-triggered methods [124, 143] where the subspace spanned by recovered subunits can be approximated by the subspace spanned by one single set of spatially shifted subunits with shared weights. Despite the similarity between our CNN models and subunit models in the literature, our systematic and comprehensive exploration of the contribution of various structural components in the CNN helps to illuminate which nonlinear components are more important to the subunit models (Section 3.5.2) and what strategies for relating subunits were more effective (Section 3.5.3).

Overall, we believe this study is the first one that systematically evaluates the relative merits of different CNN components in the context of modeling V1 neurons. We demonstrated that key components of the CNN (convolution, thresholding nonlinearity, and pooling) contributed to its superior performance in explaining V1 responses. Our results suggests that that there is a high degree of correspondence between the CNN and biological reality.

3.6.2 Complexity of V1 neural code

Using our neural dataset (Section 3.2), we have found earlier [160] that a large proportion of V1 neurons in superficial layers are selective to higher-order complex features rather than simple oriented edges. We classified these neurons selective to higher-order complex features as "higher-order" (HO) neurons and others as "orientation-tuned" (OT) neurons (Section 3.2), some of which also exhibited complex feature selectivities due to the strictness of our classification criterion [160]. In this study, we showed that the performance gap between CNN models and non-CNN models (Gabor-based models and GLMs) was more pronounced in HO neurons than in OT neurons, and we took this as an additional evidence supporting the complexity of V1 neural code in HO neurons.

In addition, by fitting intermediate features from a pre-trained, goal-driven CNN (VGG19) to our neural data, we found that a relatively high layer (`conv3_1`), which encodes relatively complex image features (Figure 3.11h), explained our neural data the best among all the VGG19 layers. This finding further reinforced the claim that V1 neurons in superficial layers might have a great degree of complex selectivity and was largely consistent with Cadena et al. [13] where the same layer (`conv3_1`) in VGG19 provides the best model of V1 responses to natural images in their study. Furthermore, the fact that such pre-trained, goal-driven neural networks performed well for explaining neural responses both in V1 (our study and Cadena et al. [13]) and in IT [89, 178, 179] is another piece of evidence that there is a high degree of correspondence between the CNN and biological reality.

3.6.3 Limitations and varieties of the CNN

While CNN models, especially those goal-driven ones pre-trained on computer vision tasks, performed very well in our study and some other studies [13] for V1 neuron modeling, we should point out that even the best-performing CNN in our study only explained about $50\,\%$ of the explainable variance in our neural data, consistent with Cadena et al. [13]. The failure of CNN models for explaining the other half of the variance in V1 data can be due to a number of reasons. First, V1 neurons are subject to network interaction and their neural responses are known to be mediated by strong long-range contextual modulation. Second, it is possible that there are some basic structural components missing in the current deep CNN methodology for fully capturing V1 neural code.

An important design issue with CNN modeling is the depth of the network. Here, we used a very basic one-convolutional-layer CNN because we can then formulate all models as one-hidden-layer neural networks (Section 3.3.4) and directly evaluate the relative contributions of different model components (Sections 3.5.2 and 3.5.3). We found that, at least for our data, adding an additional convolutional layer did not produce significant performance benefit (Supplementary Materials), and the performance difference between our baseline one-convolutional-layer CNN and the pre-trained CNN was relatively small (Figure 3.11). Our findings suggest that the true nature of V1 neurons does not need to be modeled by a very deep network. However, other studies typically use deeper CNNs. McIntosh et al. [113] used a 2-convolutional-layer CNN with to model retinal ganglion cells; Kindel et al. [78] and Cadena et al. [13] used 2- and 3-convolutional-layer CNNs to model V1 neurons respectively. It is possible that 2 or 3-layer networks are needed for modeling V1 neural responses to natural images that are richer than our artificial stimuli; in addition, higher layers in those models might be functionally equivalent to the max pooling layer in our CNN, as those multi-layer CNNs typically do not use pooling. Given there are multiple layers of neurons on the pathway between the photoreceptors in the retina and superficial layer cells in V1, biologically speaking, a much deeper CNN should provide a more accurate model in a more general setting.

Most CNN-based work models all the neurons in a data set with a single network, with shared parameters in lower layers and separate sets of parameters for different neurons in higher layers [13, 78, 81, 113]. Instead, we model each neuron separately to allow a more fair comparison between CNN models and other models (GLMs, etc.) that typically model neurons separately without parameter sharing, and a fair comparison allows us to understand the CNN's success compared to other models more conveniently. We also tried modeling all neurons using a single CNN (Supplementary Materials) with an architecture similar to that in Cadena et al. [13]; to our surprise, large single CNNs that model all neurons together performed similarly to our baseline CNNs that model each neuron separately, given roughly the same number of parameters; for example, a 3-layer single CNN with around 300k parameters trained on some (around 350) HO neurons performed similarly to separately trained CNNs, which together take around 300k parameters (883 parameters per model) as well. More investigation (better hyperparameter tuning, better network architecture, etc.) is needed to improve the performance of modeling using a single CNN on our V1 data.

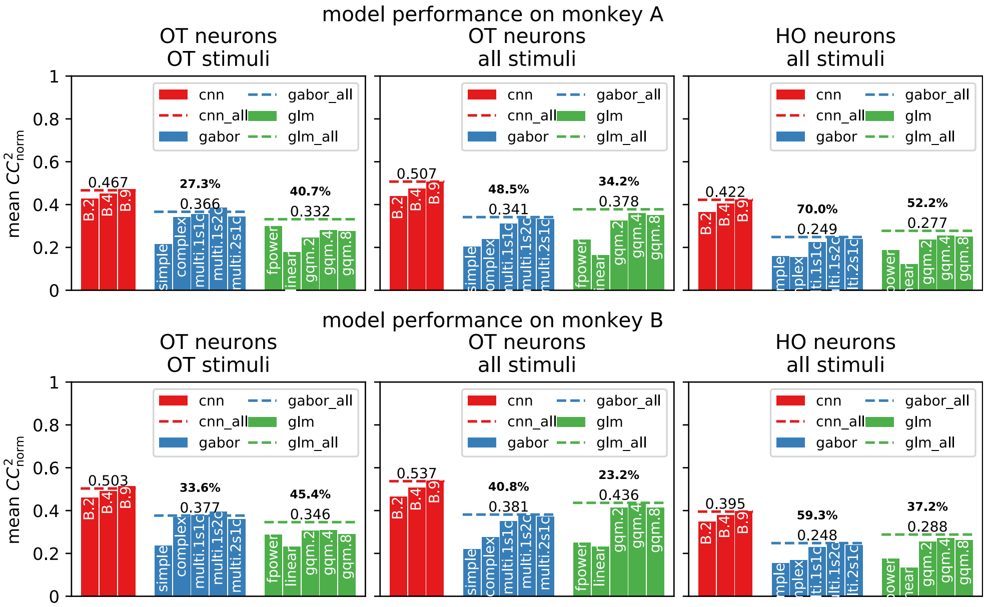


Figure 3.6: CNN models vs. others on explaining V1 neural data. Two rows of panels show results for monkey A and monkey B respectively, and three columns of panels show how models performed on different neuron subsets ("OT" and "HO"), evaluated on different subsets of stimuli ("OT" and "all"). For each panel, the model performance is shown in $\mathrm{CC}^2_{\mathrm{norm}}$ averaged over neurons in the neuron subset. For each category of models (`cnn`, `glm`, etc.), solid bars show model performance of different specific model architectures, and dashed lines (suffixed by `_all`) show the category's overall "best" performance by taking the best model architecture for each individual neuron (in terms of validation performance for CNNs and GLMs and training performance for Gabor models). Boldface numbers are the relative performance increases of the CNN classes over non-CNN classes (computed as ratios between dashed lines minus one). For CNN models (red), check Table 3.2 for their meanings. For Gabor models (blue), `complex` and `simple` mean complex cell and simple cell models; `multi.MsNc` means linear combinations of `M` simple and `N` complex model(s). For generalized linear models (green), `linear` means vanilla GLM; `fpower` means Fourier power GLM; `gqm.x` (`x` being one of `0`,`2`,`4`,`8`) means the quadratic GLM with locality `x`.

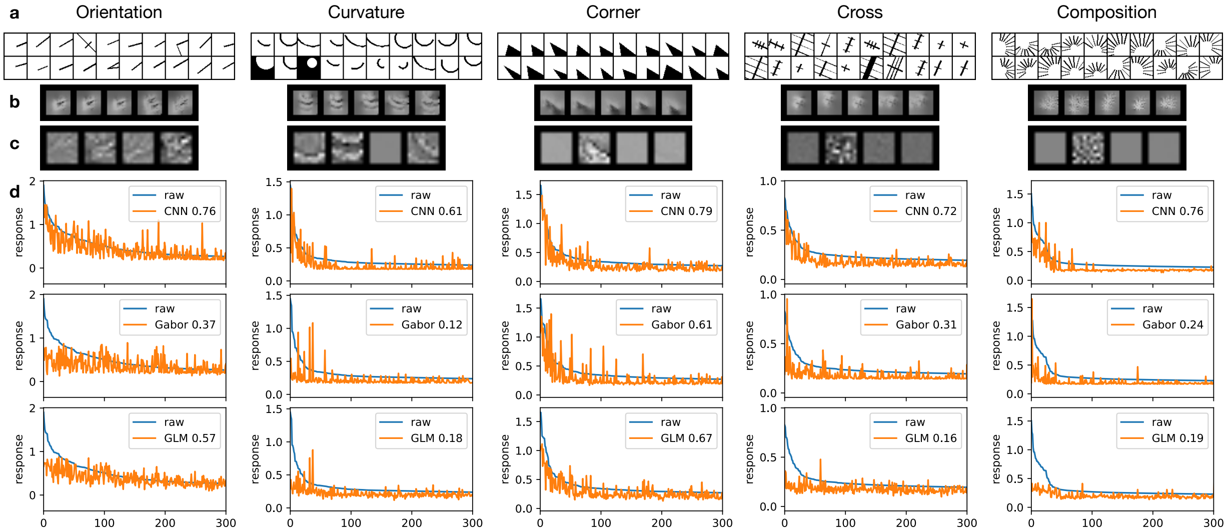


Figure 3.7: Example neurons and their fitting results. For each of the five stimulus classes shown in different columns, we show the following four pieces of information regarding the fitting of a neuron that responded better to this class than the others (**a-d**). **a** The top 20 responding stimuli of the neuron; **b** the fitted CNN fully connected output layer's visualization results (over 5 random initalizations) obtained by activation maximization [119] implemented in `keras-vis` [88]; **c** the fitted CNN's four 9 by 9 convolutional filters (each scaled by the sum of squares of its associated weights in the fully connected layer); **d** the neuron's fitting results (over testing data) on three categories of models: CNN, Gabor and GLM, with model performance in terms of $\mathrm{CC}_{\mathrm{norm}}^2$ given in the legends. As each category of models has multiple variants or architectures, we roughly speaking picked the overall best one for each category. We picked the 4-channel architecture `B.4` for CNN, as it performed almost the same as the baseline `B.9` (Figure 3.6) and allows easier visualization and interpretation; we picked `multi.1s2c` for Gabor, and `gqm.4` for GLM as they performed overall better than other variants. Check Figure 3.6 for the meanings of model names.

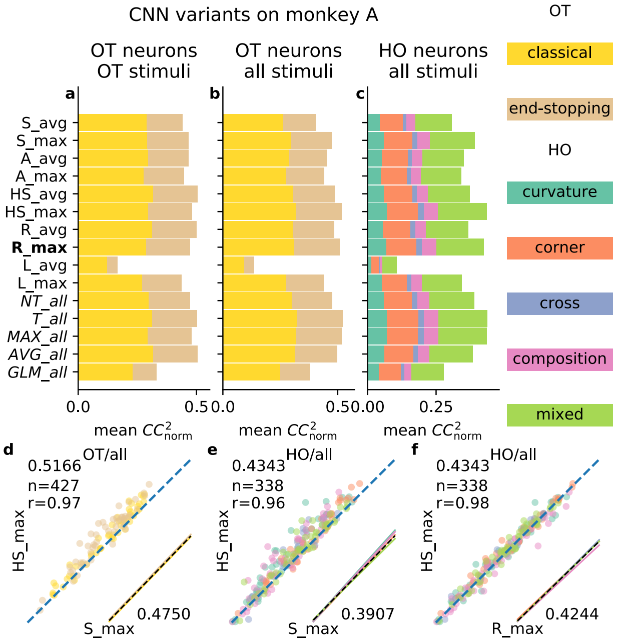


Figure 3.8: Detailed comparison of CNN variants, monkey A. **a-c** ten variants of the baseline CNN (B.9), along with the "all" model for GLMs GLM_all (Figure 3.6) for reference. In addition, four "all" CNNs, each of which constructed from CNN models with some shared structural component (thresholding nonlinearity T, non-thresholding nonlinearity NT, max pooling MAX, or average pooling AVG), are shown as well. CNN variants are named X_Y where X and Y denote nonlinearity and pooling type, respectively (Section 3.5.2). The organization of panels is the same as that in Figure 3.6, except that only results for Monkey A are shown (see Figure 3.9 for Monkey B). Rows show different models, whose performance metrics (mean $\mathrm{CC}^2_{\mathrm{norm}}$) are decomposed into components of neuron subclasses, denoted by different colors (legend on the right). For each model in some panel, the length of each colored bar is equal to the average performance over that neuron subclass multiplied by the percentage of neurons in that subclass, and the length of all bars concatenated is equal to the average performance over all neurons. The baseline model has its name in bold, and "all" models in italics. **d,e** Neuron-by-neuron comparison of the a CNN variant with thresholding nonlinearity (HS_max) vs. one without (S_max) for OT neurons, all stimuli (**d**) and HO neurons, all stimuli (**e**). For **d**, **e**, and **f**, performance metrics (mean $\mathrm{CC}^2_{\mathrm{norm}}$) are shown at corners, Pearson correlation coefficients between models are shown at the top left, and regression lines for different neuron subclasses (colored solid) together with the regression line over all neurons (black dashed) are shown at the bottom right (scaled and shifted to the corner for clarity; otherwise these regression lines will clutter the dots that represent individual neurons). **f** Comparison of two thresholding nonlinearities, for HO neurons, all stimuli. Results with max pooling are shown, and average pooling gave similar results.

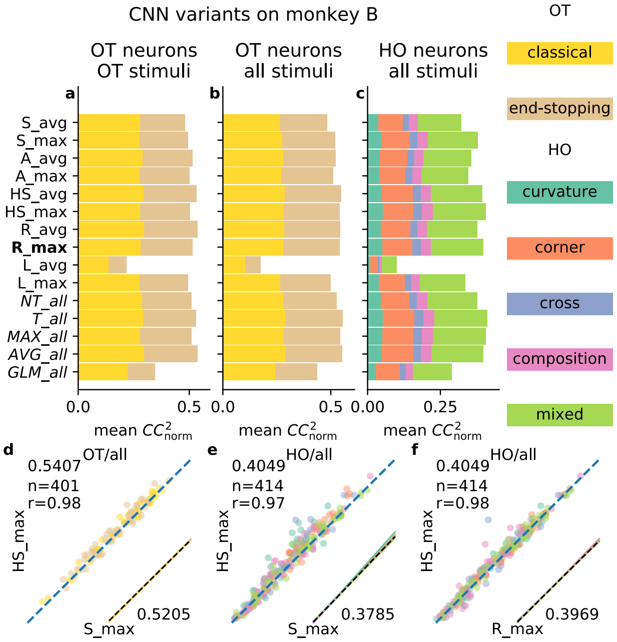


Figure 3.9: Detailed comparison of CNN variants, monkey B. Check Figure 3.8.

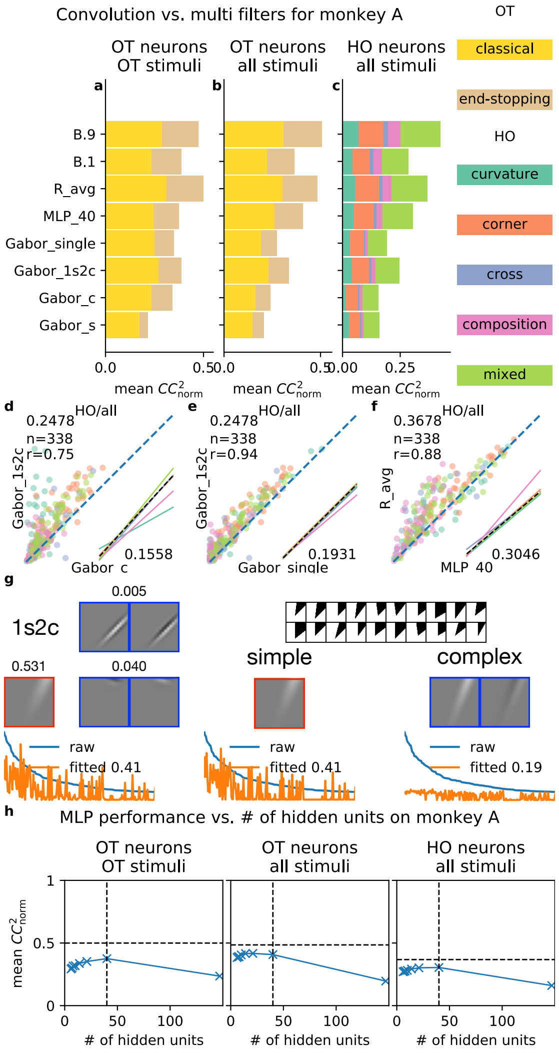


Figure 3.10: Convolution seemed more important than diverse filters. **a-f** Comparison of single- vs. multi-component Gabor models (highlighted in **d**,**e**), comparison of single- vs. multi-channel CNN models, and comparison of models with and without convolution (highlighted in **f**). See Section 3.5.3 for details. These panels have similar formats to those in Figure 3.8. **g** Learned single- (simple and complex) and multi-component (1s2c) Gabor models fitted to a particular neuron's data. This neuron was tuned to corners as shown in the top right part of the panel. For the three models (left, middle, right), we show the learned filters (top) and fitting results (bottom). Simple cell components are shown with red borders, and complex cell components are shown with blue borders. For the multi-component model, we also show the weights of different components at the top of filters. In this case, the multi-component model was dominated by its simple component with weight 0.531, which was orders of magnitude larger than the weights of its complex components. **h** Performance vs. number of hidden units for MLP models. Vertical dashed lines denote the MLP model (MLP_40) in panels **a-c**,**f**, and horizontal dashed lines show performance metrics of the CNN R_avg. Only results for monkey A are shown and monkey B gave similar results.

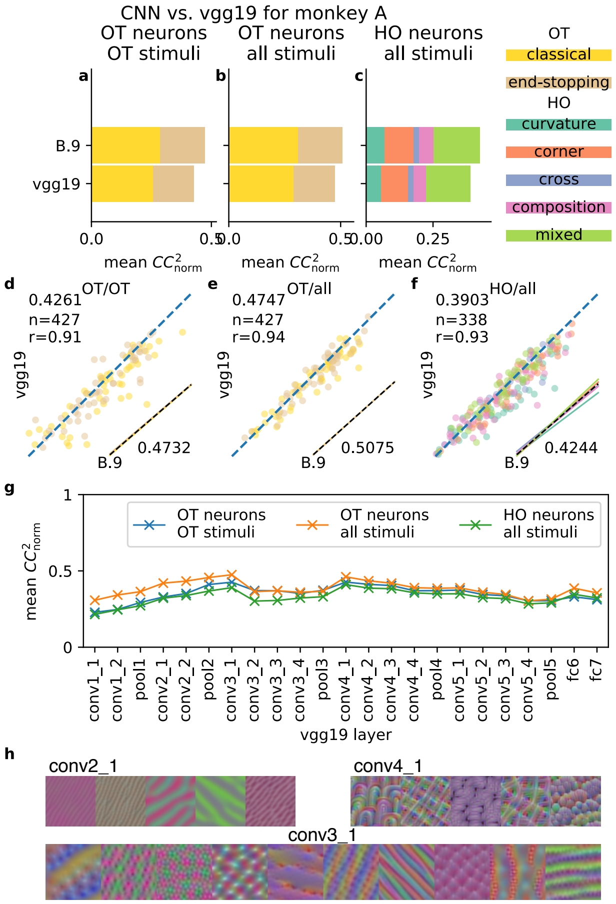


Figure 3.11: Transfer learning (goal-driven) approach for modeling V1 neurons using a pre-trained CNN (VGG19). **a-f** the best performing VGG19 layer (`conv3_1`) vs. the baseline CNN (`B.9`). These panels have similar formats to those in Figure 3.8. **g** Model performance across different VGG19 layers, for different combinations of neuron subsets and stimuli. Only results for monkey A are shown, and monkey B gave similar results (Supplementary Materials). **h** Visualization of some `conv2_1` (left), `conv3_1` (bottom), and `conv4_1` (right) units by activation maximization [88, 119]. Each unit was visualized by the image (cropped to $50\,\mathrm{px}$ by $50\,\mathrm{px}$) that maximizes the unit's activation. See Supplementary Materials for more results.

Chapter 4

Modeling neural responses of early visual areas using recurrent convolutional neural networks

Feed-forward deep neural networks are prevalent in modeling early visual areas of the brain, without accounting for the abundant recurrent connections in the visual system. We found that deep neural networks with recurrent circuits outperformed feed-forward models with matched model sizes and hyperparameters, in predicting neural responses of early visual areas to large sets of stimuli. By using a novel method that reformulates recurrent models as multi-path ensemble models, we found that the recurrent model outperformed the feed-forward model by implicitly summing over multiple feed-forward paths during inference; the multi-path ensemble inside the recurrent model allows approximating the complex function underlying recurrent biological circuits efficiently with shared parameters across paths. In addition, we found that the performance of a recurrent model was highly correlated with the the balance of short and long paths in its multi-path ensemble. Our work establishes the superiority of recurrent models in modeling neural responses of early visual areas and proposes a novel approach to understand recurrent models using multi-path ensembles, providing new understanding on the computational rationales and advantages of recurrent circuits that are so ubiquitous for biological systems.

4.1 Introduction

Feed-forward deep neural networks have been shown to be an effective model for predicting neural responses of early visual areas [13, 78, 81, 89, 179, 184]. However, abundant recurrent connections exist within each visual area and between visual areas [32, 109]. In this paper, we tried to investigate two questions. Experimentally, we'd like to know whether deep neural networks with recurrent circuits can provide a better model for predicting neural responses in the early visual areas; theoretically, we'd like to understand why recurrent models perform better from either a computational or a biological perspective, if the answer to the first question is positive.

To answer the first question, we trained a multitude of recurrent models and feed-forward models under different hyperparameters and data sets; experimental results showed that recurrent models could explain neural responses of early visual areas better than typical feed-forward models with matched hyperparameters and model sizes, especially when there was less training data.

To answer the second question, we developed a novel method to reformulate a recurrent model as an ensemble of feed-forward models. Our novel method is based on the hypothesis that the advantage of the recurrent model rests on the ensemble of multiple feed-forward paths embedded in the recurrent computation and such multitude of paths makes the recurrent model more flexible compared to a feed-forward model. By reformulating the recurrent model as a multi-path model and analyzing the recurrent model through its multi-path ensemble, we can (1) compare recurrent models and feed-forward models in a unified framework where feed-forward models are degenerate multi-path models with one path; (2) understand why recurrent models can outperform feed-forward models by comparing their multi-path ensembles; (3) understand the conditions under which recurrent models performed better by comparing the multi-path ensembles of different recurrent models and by performing ablation studies on the ensembles; (4) connect recurrent models to the existing literature on ensemble models in the machine learning community.

By studying recurrent models and feed-forward models via their corresponding multi-path ensembles, we found that the recurrent model outperformed the feed-forward one due to the former's compact and implicit multi-path ensemble that allows approximating the complex function underlying recurrent biological circuits with efficiency. In addition, we found that the performance differences among the recurrent models we explored were highly correlated with the differences in their multi-path ensembles; in particular, models with more relative weights on shorter paths tended to perform better than models with more relative weights on longer paths.

Our work establishes that the recurrent model than the purely feed-forward model for predicting neural responses in the early visual areas, complementing previous studies on feed-forward models [13, 78, 81, 179] and consistent with very recent studies on recurrent ones [77, 155]. Our most interesting contribution is to establish that the superiority of the recurrent model for neural prediction can be attributed to the implicit and compact multi-path ensemble inside the model, and that a balance of different paths in the ensemble is necessary for the model to achieve the best performance. This work provides new understanding on the computational rationales and advantages of recurrent circuits that are ubiquitous in biological systems [32].

4.2 Related work

4.2.1 Modeling visual areas of the brain using neural network models with recurrent circuits

In the brain, it is well known that there are local horizontal recurrent connections between neurons in the same area [73] and long-range feedback recurrent connections between neurons in different areas [32]. There are at least three types of studies on recurrent circuits in visual areas.

The first type of studies focuses on modeling various neural response properties (surround suppression, end-stopping, etc.) thought be to related to recurrent circuits [18, 19, 20, 134, 156, 157, 158, 180, 185] but these models' predictive power on arbitrary input stimuli are either unknown or worse than deep feed-forward networks that have recently become popular in the visual neuroscience community for predicting neural responses to arbitrary stimuli [13, 78, 81, 89, 179].

The second type of studies focuses on showing the advantage of recurrent circuits in modeling temporal dynamics of visual neurons [77, 93, 116] or human behaviors in visual tasks [155]. Our work differs from these studies by training recurrent models to predict average firing rates instead of temporal dynamics of visual neurons. In some sense, the baseline feed-forward models in the previous neural modeling studies were bound to perform worse because feed-forward models inherently cannot model temporal dynamics; instead, our study demonstrated the superiority of recurrent models in a setting where feed-forward models are more likely to perform well.

The third type of studies uses recurrent circuits in the brain as a motivation to compare recurrent models and feed-forward models for solving traditional computer vision tasks such as image classification [93, 116, 155]. Our work complements these studies by studying the advantage of recurrent models over feed-forward models in neural modeling tasks [13, 78, 81] instead of computer vision ones.

4.2.2 Multi-path ensemble models and their relationships with recurrent models

Multi-path models have been widely studied in the computer vision community; highway networks [159], ResNet [50, 51], FractalNet [94], and DenseNet [61] are among the most representative ones. Many multi-path models can be conceptually understood as ensembles summing over smaller networks [167].

There have been relatively few studies connecting recurrent models to multi-path ensemble models in the deep learning context. For ResNets, Liao and Poggio [104] have pointed out the equivalence between a specific type of recurrent networks with weight-tied ResNets, and Chen et al. [16] formulate deep weight-tied ResNets as approximating the dynamics of continuous recurrent models specified in ordinary differential equations. To the best of our knowledge, there is no work explicitly connecting recurrent models to multi-path ensemble models in general. The most similar work to ours is probably Zhang et al. [181], which characterizes recurrent models in terms of some architectural complexity measures (feed-forward depth, recurrent depth, and recurrent skip coeffcient) motivated by graph theory and quantified the relationship between performance and these complexity measures.

In the deep learning community, certain types of recurrent models have been analyzed in other ways. Bai et al. [5] have found that weight-tied deep recurrent networks converge to a fixed point of the recurrent dynamics, and Hinton et al. [57] have proved that an infinite logistic belief net with tied weights is equivalent to a restricted Boltzmann machine [55].

4.3 Methods

As stated in Section 4.1, there are two questions we try to answer in this study. Experimentally, we'd like to know whether deep neural networks with recurrent circuits can provide a better model for predicting neural responses in early visual areas; theoretically, we'd like to understand why recurrent models perform better from either a computational or a biological perspective, if the answer to the first question is positive. To answer the first question, we trained tens of thousands of recurrent models and feed-forward models with different hyperparameters using different data sets; the architectures of our recurrent and feed-forward models are described in Section 4.3.1, the data sets in Section 4.3.2, and other details in Section 4.3.3. To answer the second question, we developed a novel method to reformulate a recurrent model as an ensemble of feed-forward models, with details given in Section 4.3.4.

4.3.1 Models

4.3.1.1 Baseline feed-forward models

Our baseline feed-forward models are based on those in Cadena et al. [13], Klindt et al. [81] (Figure 4.1a) which are state of the art in predicting neural responses in early visual areas to our knowledge. We trained many baseline feed-forward models of different model sizes and other hyperparameters (Section 4.3.3.2), and each baseline model is a feed-forward model with some convolutional layers (Figure 4.1a). During inference, the information about the static input stimulus flows from one layer to the next layer in sequence. In particular, input image pixels are first passed through a batch-normalization (BN) layer [67] and then a number of convolutional processing blocks (CPBs) each of which implements the ubiquitously observed linear-nonlinear operation [15, 136] using convolution, BN, and activation layers (Figure 4.1d). Afterwards, the output feature map of the last processing block is passed into an average-pooling layer whose output is passed into a factorized fully-connected (or linear) layer [81]. Finally, the output of the linear layer is passed through an activation layer to generate the predicted neural responses. Following previous studies [13, 81] and as well as based on our pilot experiments, the first convolutional layer has a relatively large kernel size (9) and subsequent layers have a small kernel size of 3. More details on the explored hyperparameters (number of channels, type of activation function, etc.) can be found in Section 4.3.3.2, and a more detailed description of model inference can be found in Appendix B.2.1.

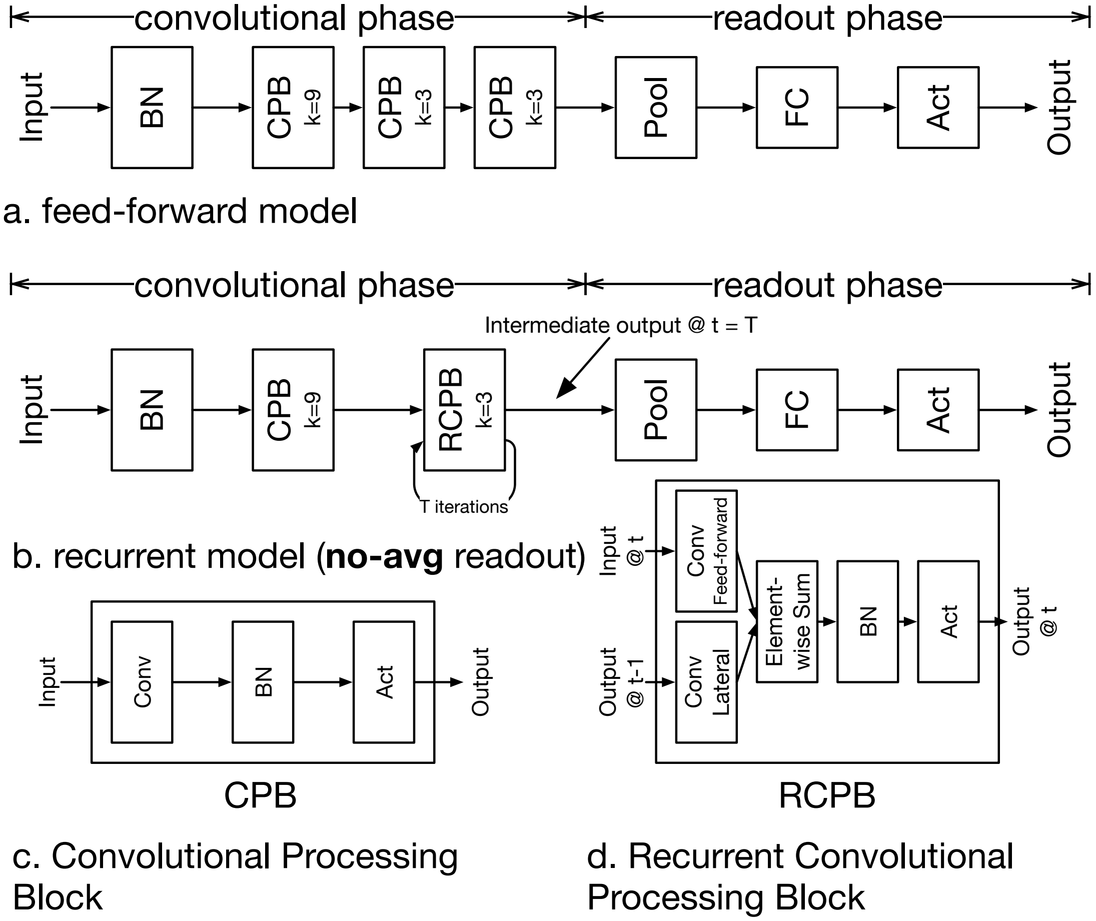


Figure 4.1: Models explored in this study. **(a)** shows the architecture of an example baseline feed-forward model. **(b)** show the architecture of an example recurrent model under `no-avg` readout mode (Section 4.3.1.2.2). For both feed-forward and recurrent models, the model inference can be divided into two phases as shown at the top of **(a)** and **(b)**: the convolutional phase and the readout phase. The concept of inference phases will be used in Section 4.3.1.2. **(c)**, **(d)** show the internal architectures of a convolutional processing block (`CPB`) and a recurrent convolutional processing block (`RCPB`), respectively. The kernel size of a (R)CPB is denoted by `k` in **(a)**,**(b)**. In this figure, we have in total three convolutional blocks for the feed-forward model **(a)** but only two blocks for the recurrent one **(b)** because a RCPB equals two CPBs in terms of model size. See Sections 4.3.1.1, 4.3.1.2 for details.

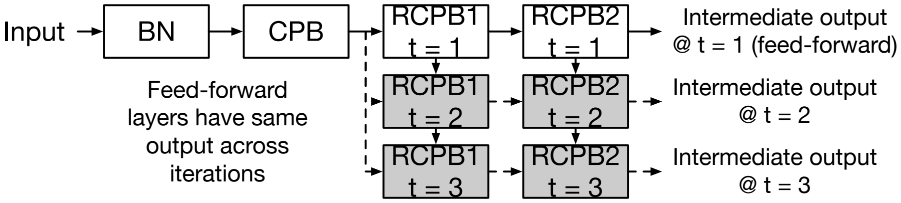

Input
BN
CPB
RCPB1 t = 1
RCPB2 t = 1
Intermediate output @ t = 1 (feed-forward)
RCPB1 t = 2
RCPB2 t = 2
Intermediate output @ t = 2
RCPB1 t = 3
RCPB2 t = 3
Intermediate output @ t = 3
Feed-forward layers have same output across iterations

Figure 4.2: Information flow in the first half of the model inference process for a recurrent model. The example model has one CPB, two RCPBs, and three iterations in total. In the first iteration $t = 1$, the information flow is the same as that in a baseline feed-forward model. In later iterations, the output of a RCPB depends on both the layer below at the current iteration as well as the RCPB's own output at the previous iteration. The extra information flow of a recurrent model relative to a feed-forward one is shown in shaded blocks and dashed lines.

4.3.1.2 Recurrent models

We explored the simplest possible recurrent variants [155] of our baseline feed-forward models. As shown in Figure 4.1b, the general architecture of a recurrent model is similar to that of a baseline feed-forward model, with two differences. First, all but the first CPBs are replaced by recurrent convolutional processing blocks (RCPBs) which generate different intermediate outputs across different iterations; second, the intermediate outputs of the highest convolutional block across different iterations can be combined in different ways to generate the final output. These two changes take place in the first phase ("convolutional phase" on top of Figures 4.1a,b) and the second phase ("readout phase") of the model inference process, respectively.

4.3.1.2.1 RCPB instead of CPB In the first phase of the inference process ("convolutional phase" on top of Figures 4.1a,b), a recurrent model replaces all but the first CPBs in a baseline feed-forward models with recurrent convolutional processing blocks (RCPBs). During model inference, information flows over each CPB only once whereas it flows over each RCPBs for multiple iterations. The detailed information flow for a recurrent model across iterations are illustrated in Figure 4.2. The total number of iterations is a model hyperparameter. When the total number of iterations is 1, a recurrent model is equivalent to a feed-forward model.

The general architecture of Recurrent Convolutional Processing Blocks (RCPBs) is the same as that of the recurrent processing blocks in Spoerer et al. [155]. A regular convolutional processing block (CPB) passes its input in sequence through a convolutional layer, a batch normalization layer, and an activation layer (Figure 4.1c). A recurrent convolutional processing block (RCPB) extends a CPB by having two inputs in the convolutional stage, one from the output from the *previous* stage of the model in the *current* iteration and the other from the output of the *current* RCPB in the *previous* iteration. As shown in Figure 4.1d, the two inputs are passed through separate convolutional layers (`Conv Feed-forward` and `Conv Lateral`) and the two layers' outputs are summed together element-wise as the input to the batch normalization layer. Note that while the convolutional layers (`Conv Feed-forward` and `Conv Lateral`) stay the same across different iterations during inference, the batch normalization layers are learned separately for each inference iteration, following Spoerer et al. [155]. Separately learning batch normalization layers makes the learning much easier and introduces negligible increase in model size (Sections B.2.2,4.5.3.1).

4.3.1.2.2 Combination of intermediate outputs for the final output For a model with T iterations in total, the first phase of model inference yields T intermediate outputs (Figures 4.1b,4.2) from the highest convolutional block. In the second phase ("readout phase") of model inference, a recurrent model in general can make use of these intermediate outputs in different ways to generate (read out) the final model output. There are multiple possibilities to combine these outputs to generate the final predicted neural responses, and different possibilities combine and utilize the different information flows across iterations (Figure 4.2) differently. We explored four different ways to generate neural response prediction from the intermediate outputs, or four different readout modes, as shown in Figures 4.3,4.1b and described as follows. The readout modes are named based on how they average the intermediate outputs across iterations. As we will show later in Section 4.4, the performance differences among recurrent models with different readout modes (Section 4.4.1) can be explained by the readout modes' different weighting schemes on intermediate outputs (Sections 4.4.2,4.4.3).

1. `no-avg` (Figure 4.1b) No averaging. The intermediate output at the last iteration is passed through the remaining layers of the model to get the final output.
2. `early-avg` (Figure 4.3a) Early averaging only. It's similar to `no-avg` except that the average of intermediate outputs across iterations is passed through the remaining layers of the model.
3. `late-avg` (Figure 4.3b) Late averaging only. The neural response predictions based on the individual intermediate outputs across iterations are averaged as the final output.
4. `2-avg` (Figure 4.3c) Both early and late averaging. First the cumulative averages of intermediate outputs across iterations are computed and then the neural response predictions based on these cumulative averages are further averaged as the final output.

Based our pilot experiments and for simplicity in matching the model size between recurrent and feed-forward models (Section 4.3.3.2.2), a recurrent model does not involve recurrent computation in its first CPB, which has a large kernel size (Section 4.3.1.1), and only adds recurrent computation to subsequent CPBs. To match the model size between a feed-forward model and a recurrent one, every two CPBs of kernel size 3 in a feed-forward model is replaced by one RCPB of kernel size 3 in corresponding recurrent models. For a more detailed description of model inference, see Appendix B.2.2.

4.3.2 Data sets

We collected neural responses of early visual areas (V1, V2) to two sets of static images, under two different stimulus presentation paradigms (Figure 4.4) that have been commonly used in the literature [13, 20]. In the ImageNet 8K data set, each stimulus was presented for a very short period ($47\,\mathrm{ms}$), and in the NS 2250 data set, each stimulus was presented for a longer period ($500\,\mathrm{ms}$). More details can be found in Appendix B.1.

4.3.2.1 ImageNet 8K

This data set contains responses of 79 neurons to 8000 images from ImageNet [141], following a stimulus presentation paradigm similar to that in Cadena et al. [13]. In each trial (Figure 4.4a), we showed a random sequence of 16 images and each image lasted for about $47\,\mathrm{ms}$. Overall six trials were collected for each image per neuron.

4.3.2.2 NS 2250

This data set contains responses of 34 neurons to the 2250 images used in Tang et al. [161]. In each trial (Figure 4.4b), we presented each image separately for $500\,\mathrm{ms}$. Overall, 8 to 10 trials were collected for each image per neuron.

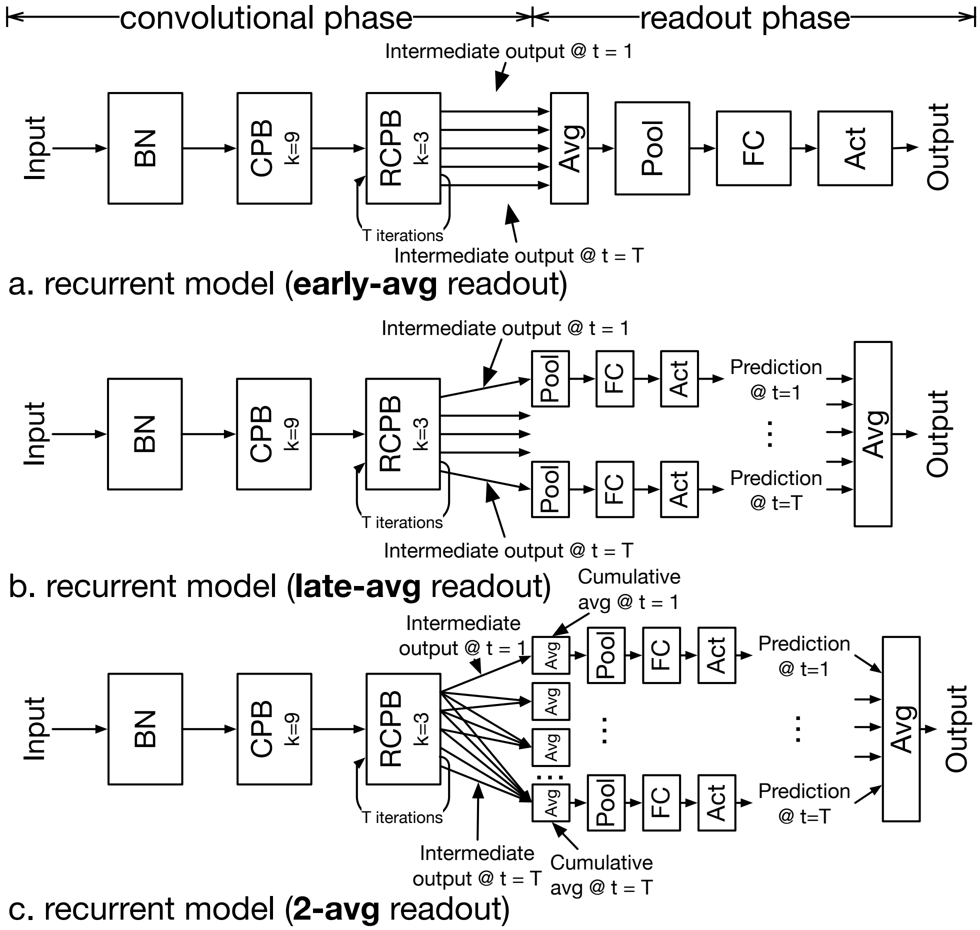
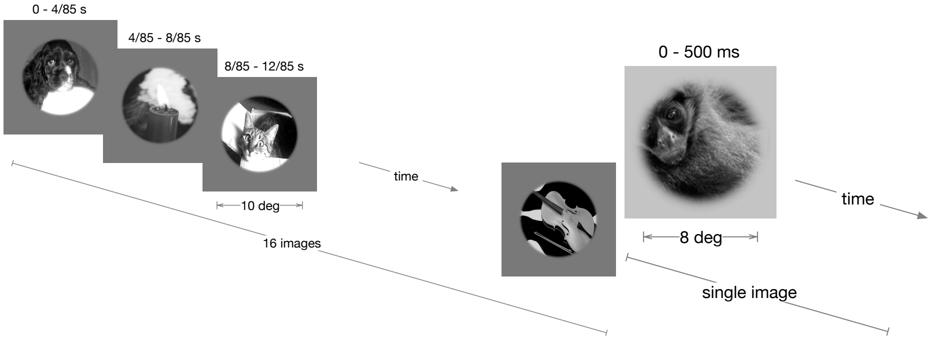


Figure 4.3: Readout modes `early-avg` (**a**), `late-avg` (**b**), and `2-avg` (**c**) explored for recurrent models in this study. We explored four readout modes in total, with the simplest one (`no-avg`) shown earlier in Figure 4.1b. The style of the figure follows that in Figure 4.1. Same as Figure 4.1b, the inference for a recurrent model can be divided into two phases, the convolutional phase and the readout phase are denoted at the top of the whole figure. All models shown in this figure have the same convolutional phase but different readout phases.



(a) (b)

Figure 4.4: Stimulus presentation paradigms of a single trial, for ImageNet 8K **(a)** and NS 2250 **(b)**. The style of the figure is adapted from Cadena et al. [13].

4.3.3 Implementation details

4.3.3.1 Data preprocessing

Following practices in the literature [13, 81], for each data set, we used cropped and downsampled images as the input and normalized average neural responses as the output for modeling purposes. See Appendix B.3 for full details.

4.3.3.2 Hyperparameters

To comprehensively evaluate recurrent models vs. feed-forward models, we trained and tested multiple variants of recurrent and feed-forward models with different hyperparameters. Note that while hyperparameters typically only refer to configurations that affect model architecture, here we use the word more generally to denote all configurations that can affect model performance, including amount of training data, random seed, loss function, and activation layer.

Table 4.1: Hyperparameters explored for feed-forward models. k denotes the kernel size for a convolutional block

category	name	values explored
training data hyperparameters	amount of training data	25%, 50%, 100% (1280, 2560, 5120 for ImageNet 8K, 350, 700, 1400 for NS 2250)
model size-related model hyperparameters	# of convolutional blocks	3 (1 CPB @ k=9 + 2 CPBs @ k=3), 5 (1 CPB @ k=9 + 4 CPBs @ k=3)
	# of channels per convolutional layer	8, 16, 32, 48, 64 for ImageNet 8K, 8, 16, 32 for NS 2250
model size-independent model hyperparameters and loss functions	loss function	mean squared error, mean Poisson loss
	activation layer	ReLU, softplus
	order of BN and activation in the first convolutional block	BN before act, BN after act
randomness hyperparameters	model initialization seed	0, 1

4.3.3.2.1 Feed-forward models We explored feedforward models under all combinations of hyperparameters listed in Table 4.1. In total, on the ImageNet 8K data we trained in total 480 feed-forward models with different hyperparameters, and on the NS 2250 data we trained in total 288 models. These hyperparameters are classified into four categories.

- **Training data** and **model size-related model hyperparameters**. we explored hyperparameters in these two categories because we feel training data amount and model size affect the relative performance difference between feedforward and recurrent models. In particular, we hypothesized that the additional circuit priors from recurrent models would be more useful when there was less training data and the model size was larger.

- **Model size-independent model hyperparameters and loss functions** We explored hyperparameters in this category because we wanted to know if the advantage of recurrent models over feed-forward models is limited to certain choices of model size-independent model hyperparameters and loss functions, or if such advantage is universal across choices of model size-independent hyperparameters and loss functions.
- **Randomness hyperparameters** We explored different model initialization seeds because we wanted to obtain robust metric numbers for our models by averaging results over different seeds. Ideally we should also explore over other randomness hyperparameters, such as the seed used to split data into training, validation, and testing sets. We did not do so due to computation resource constraints.

4.3.3.2.2 Recurrent models For each feed-forward model, we trained 24 corresponding recurrent models of the same size. To keep the comparison as fair as possible, each of the 24 recurrent models have the same hyperparameters as the feed-forward one on those hyperparameters listed in Table 4.1, except with a decreased number of convolutional layers to match the model size because a recurrent convolutional processing block has roughly twice as many parameters as a convolutional processing block with the same number of channels; therefore, when the feed-forward model has three layers (1 CPB of kernel size 9 + 2 CPBs of kernel size 3), the corresponding recurrent models have two layers (1 CPB of kernel size 9 + 1 RCPB of kernel size 3, with one RCPB equal to two CPBs in model size); when the feed-forward model has five layers (1 CPB of kernel size 9 + 4 CPBs of kernel size 3), the corresponding recurrent models have three layers (1 CPB of kernel size 9 + 2 RCPBs of kernel size 3). In addition, the 24 recurrent models differ in their recurrence-related model hyperparameters listed as follows.

- **Number of iterations**, with six possible values 2 to 7. This hyperparameter affects the amount of recurrent computation during model training and inference. We wanted to explore this hyperparameter because obviously amount of recurrence may affect model performance. Note that we always use the same number of iterations during training and testing; e.g. a recurrent model with T iterations during training is always evaluated with T iterations during testing.
- **Readout mode**, with four possible values (`no-avg`, `early-avg`, `late-avg`, `2-avg`; Section 4.3.1.2.2). We want to explore this hyperparameter because we wanted to know if some readout modes perform better than others and if there exists some optimal way to make use of intermediate outputs generated by a recurrent model.

In total, for the ImageNet 8K data set, we traind 480 feed-forward models and $11\,520$ recurrent models; for the NS 2250 data set, we trained 288 feed-forward models and 6912 recurrent models; we trained fewer models for the NS 2250 data set because the NS 2250 data set is much smaller than the ImageNet 8K.

4.3.3.2.3 Other hyperparameters As mentioned in the end of Section 4.3.1.1, In either of a feed-forward model or a recurrent model, the first convolutional layer has a kernel size of 9 and later convolutional layers all have a kernel size of 3. The pooling layer always has a kernel size of 3 and a stride of 3.

4.3.3.3 Model optimization

We trained all models using PyTorch [126] and followed Klindt et al. [81] for model optimization.

4.3.3.3.1 Objective function For each model, the objective function to be minimized was the sum of two parts.

Neural prediction loss Depending on the choice of hyperparameters (Section 4.3.3.2), the loss can be either mean squared loss or mean Possion loss between between the predicted neural response and recorded ground truth averaged across neurons and images. Note that for recurrent models with `late-avg` or `2-avg` readout modes, following the practice in Spoerer et al. [155], we computed the neural prediction loss by averaging the losses over individual iterations.

Regularization terms We applied L1 sparsity penalties as regularization; in addition, for the first convolutional layer of the model, we applied smoothness regularization as in Section 5.2 of Klindt et al. [81].

4.3.3.3.2 Optimization algorithm Given the objective function, we used the same optimization algorithm as in the public code of Klindt et al. [81]; in particular model parameters were optimized sequentially in three phases, by three Adam optimizers with decreasing learning rates. Early stopping was applied in each phase, guided by the neural prediction loss evaluated on the validation set. For each data set, roughly $64\,\%$,16 $\%$,and $20\,\%$ of images were used for training, validation, and testing, respectively. Due to computation resource constraints, we only created one particular split of training, validation, and testing sets for each of ImageNet 8K and NS 2250. To make sure our conclusions were not dependent on particular data splits, we retrained a subset of models using another data split and found similar results (Appendix B.10).

4.3.3.4 Model performance evaluation

Given a trained model, we use average $\mathrm{CC}^2_{\text{norm}}$ [149, 184] over all neurons to quantify its performance on a data set. For each neuron, we compute its $\mathrm{CC}^2_{\text{norm}}$ based on Eqs. (4.1).

$$\mathrm{CC}^2_{\text{norm}} = \frac{\mathrm{CC}^2_{\text{raw}}}{\mathrm{CC}^2_{\text{max}}} \tag{4.1a}$$

$$\mathrm{CC}_{\text{raw}} = \mathrm{Pearson}(\vec{\boldsymbol{r}}, \vec{\hat{\boldsymbol{r}}}) \tag{4.1b}$$

Conceptually, $\mathrm{CC}^2_{\text{norm}}$ measures the amount of fraction of explained variance, with trial-to-trial variance in neural responses discounted. To get the $\mathrm{CC}^2_{\text{norm}}$ for a neuron, we first compute the raw Pearson correlation $\mathrm{CC}_{\mathrm{raw}}$ between the ground truth trial-averaged neural responses $\vec{\boldsymbol{r}}$ and the model responses $\hat{\vec{\boldsymbol{r}}}$, then we divide $\mathrm{CC}_{\mathrm{raw}}$ by $\mathrm{CC}_{\mathrm{max}}$, which is defined in Eq. (B.3c) of Appendix B.4 and estimates the maximal Pearson correlation coefficient an ideal model can achieve given the noise in the neural data [60, 149]. See Appendix B.4 for details. While the derivation of $\mathrm{CC}_{\mathrm{max}}$ is technically complicated, it's positively related with the Pearson correlation between mean responses computed over half of trials and those computed over the other half [60, 149]. We use $\mathrm{CC}^2_{\mathrm{norm}}$ instead of raw $\mathrm{CC}^2_{\mathrm{raw}}$ to quantify model performance because the former is more robust to trial to trial response fluctuations. Our main results in Section 4.4.1 still held if we measure the performance in $\mathrm{CC}^2_{\mathrm{raw}}$. Note that the trial-to-trial neural response data from the ImageNet 8K data were less stable compared to those from the NS 2250 data, resulting in a lower $\mathrm{CC}_{\mathrm{max}}$ for ImageNet 8K. In turn, due to the normalization effects of $\mathrm{CC}^2_{\mathrm{max}}$ on the raw squared Pearson correlation $\mathrm{CC}^2_{\mathrm{raw}}$, the model performance metrics to be presented in Section 4.4 were generally higher on ImageNet 8K than NS 2250, although the performance metrics on ImageNet 8K were lower than NS 2250 if measured in $\mathrm{CC}^2_{\mathrm{raw}}$.

4.3.3.5 Code and data

The data and code to reproduce the results can be found at `https://github.com/leelabcnbc/thesis-yimeng-v2`.

4.3.4 Reformulation of recurrent models as multi-path ensembles

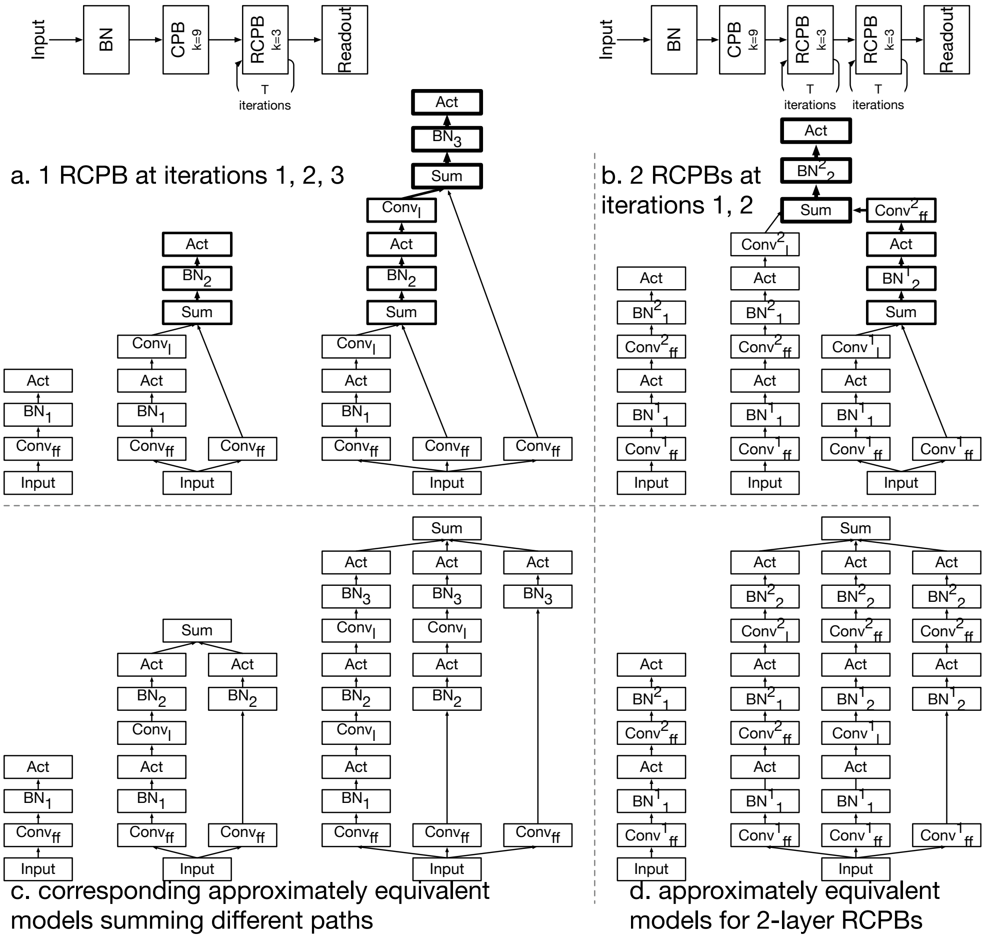
To better understand why recurrent models outperformed feed-forward ones, and to better understand the relationships among different recurrent models of different readout modes, we tried to reformulate each recurrent model as a multi-path model that is easier to analyze, inspired by Veit et al. [167]. In particular, the reformulation converts the recurrent model's recurrent processing blocks (RCPBs) into a approximately equivalent multi-path ensemble with multiple feed-forward paths. Figure 4.5 illustrates the reformulation process. During model inference, recurrent blocks compute responses in an iterative fashion, resulting in multiple information flow paths of different lengths with shared components such as convolutional and batch normalization layers (Figure 4.5a,b). To make the analysis of recurrent computation easier, we can approximate the actual information flow as summations of simpler feed-forward paths (Figure 4.5c,d), separating shared components into separate individual paths. The resulting multi-path ensemble is a summation of multiple feed-forward paths with shared parameters.

While the reformulation is approximate in nature as we ignored the effects of activation functions and bias parameters in the derivation, in practice we found that the reformulation largely held as to be shown in Section 4.4.2.

In short, through the above multi-path reformulation, we can convert each recurrent model into a corresponding model that contains a multi-path ensemble with shared parameters across the feed-forward paths; the conversion can help to deepen our understanding of recurrent networks in the following ways as mentioned in Section 4.1.

1. Comparing all feed-forward and recurrent models we trained in a unified framework, as feed-forward models can be thought of as having a degenerate multi-path ensemble with a single path.

2. Understanding why recurrent models can outperform feed-forward models under different hyperparameters and amounts of training data by comparing their multi-path ensembles.
3. Understanding the conditions under which recurrent models performed better by comparing the multi-path ensembles of different recurrent models and by performing ablation studies on the ensembles.
4. Connecting recurrent models to the existing literature on ensemble models in the machine learning community.



a. 1 RCPB at iterations 1, 2, 3

b. 2 RCPBs at iterations 1, 2

c. corresponding approximately equivalent models summing different paths

d. approximately equivalent models for 2-layer RCPBs

Figure 4.5: Recurrent computation approximately understood as summation of feed-forward chains of different depths. **(a)** shows the information flows of a 1-layer RCPB at iterations 1, 2, and 3 during model inference, and **(b)** shows the information flows of a 2-layer RCPB stack at iterations 1 and 2; the width of black lines is proportional to the number of information flows passing between adjacent components and the width of a component's boarder is proportional to the the number of information flows passing through it. **(c)** and **(d)** show the information flows of corresponding multi-path models and each flow sums over feed-forward chains of different depths.

4.4 Results

4.4.1 Recurrent models outperformed similarly-sized feed-forward models

In this section, we present the performance metrics of recurrent models vs. feed-forward ones. As described in Section 4.3.3.2, we trained a huge number of models with different architectures, hyperparameters, and amounts of training data. To systematically compare recurrent and feed-forward models, we will present the results at different aggregation levels.

- First in Section 4.4.1.1 we directly compare individual recurrent and feed-forward models without any aggregation, except for model initialization seed (Section 4.3.3.2, "randomness hyperparameters"). We performed the aggregation to obtain more robust and reliable model performance metrics
- Then in Section 4.4.1.2 present results under every combination of model size and amount of training data, aggregating over hyperparameters that do not affect model size such as loss function and activation layer (Section 4.3.3.2, "model size-independent hyperparameters and loss functions"). We decided to aggregate over these hyperparameters because empirically the relative performance change of recurrent over feed-forward ones did not change much w.r.t. these hyperparameters.
- Finally in Section 4.4.1.3 we summarize the results by additionally aggregating over number of channels, or aggregating over number of channels as well as number of layers (Section 4.3.3.2, "model size-related hyperparameters"), to get a high level summary of recurrent vs. feed-forward models, as recurrent models consistently outperformed feed-forward models under different model sizes.

4.4.1.1 Individual recurrent models outperformed feed-forward models with same hyperparameters and matched size

Figure 4.6 compares individual feed-forward models with recurrent ones, under each combination of readout mode and number of iterations (Section 4.3.3.2.2), on the ImageNet 8K data set. Recurrent models consistently outperformed feed-forward ones, except for a few cases, such as `no-avg` readout mode combined with a high number of iterations. Similar results held on the NS 2250 data set (Figure 4.7).

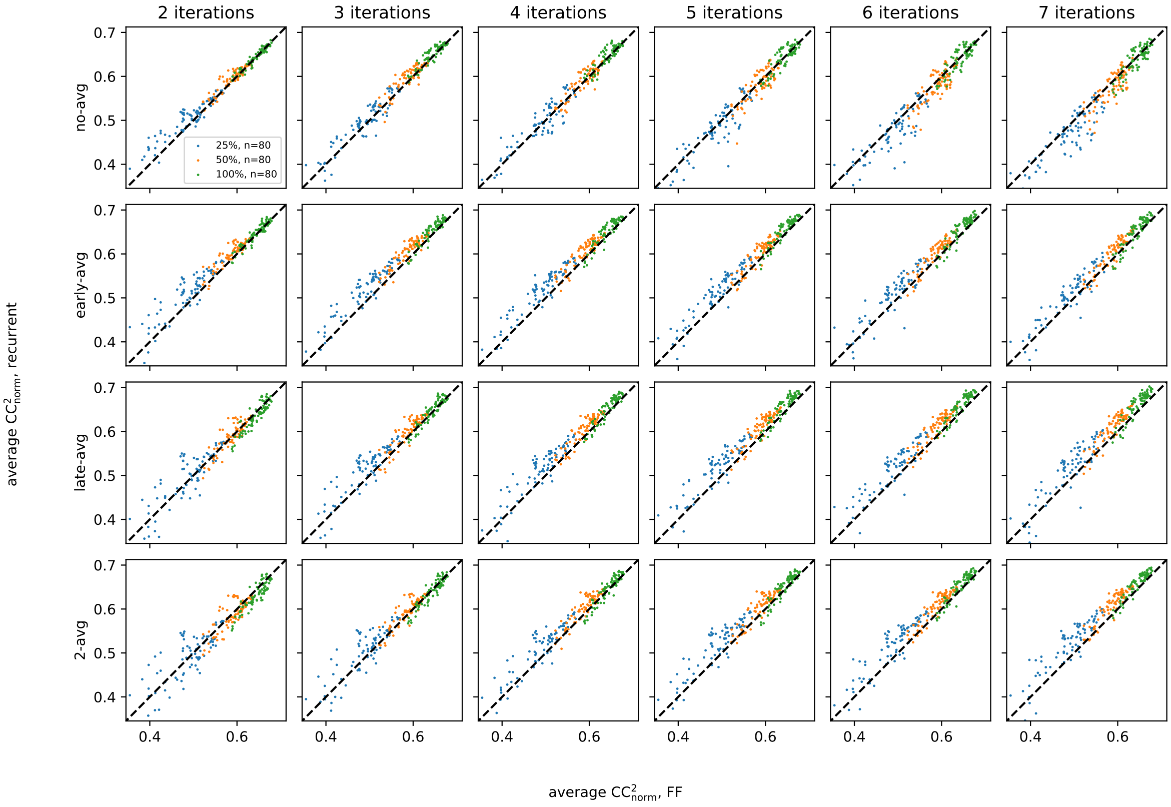


Figure 4.6: Recurrence models vs. similarly-sized feed-forward models under 24 combinations of readout mode and number of iterations, for ImageNet 8K data. Panels in the same row have the same readout mode as indicated on the left of the whole figure, and panels in the same column have the same number of iterations as indicated on the top of the whole figure. While we trained 160 feed-forward models and their corresponding recurrent models under 24 conditions (Section 4.3.3.2) for each explored training data size, each panel only shows $n = 80$ pairs of (aggregated) recurrent vs. feed-forward models per training data size, as we averaged the results over the two explored model parameter initialization seeds for more robust metrics (Section 4.3.3.2). In each panel, model pairs trained under different amounts of training data (25%, 50%, 100%) are shown in different colors, with 80 (aggregated) model pairs for each explored training data size.

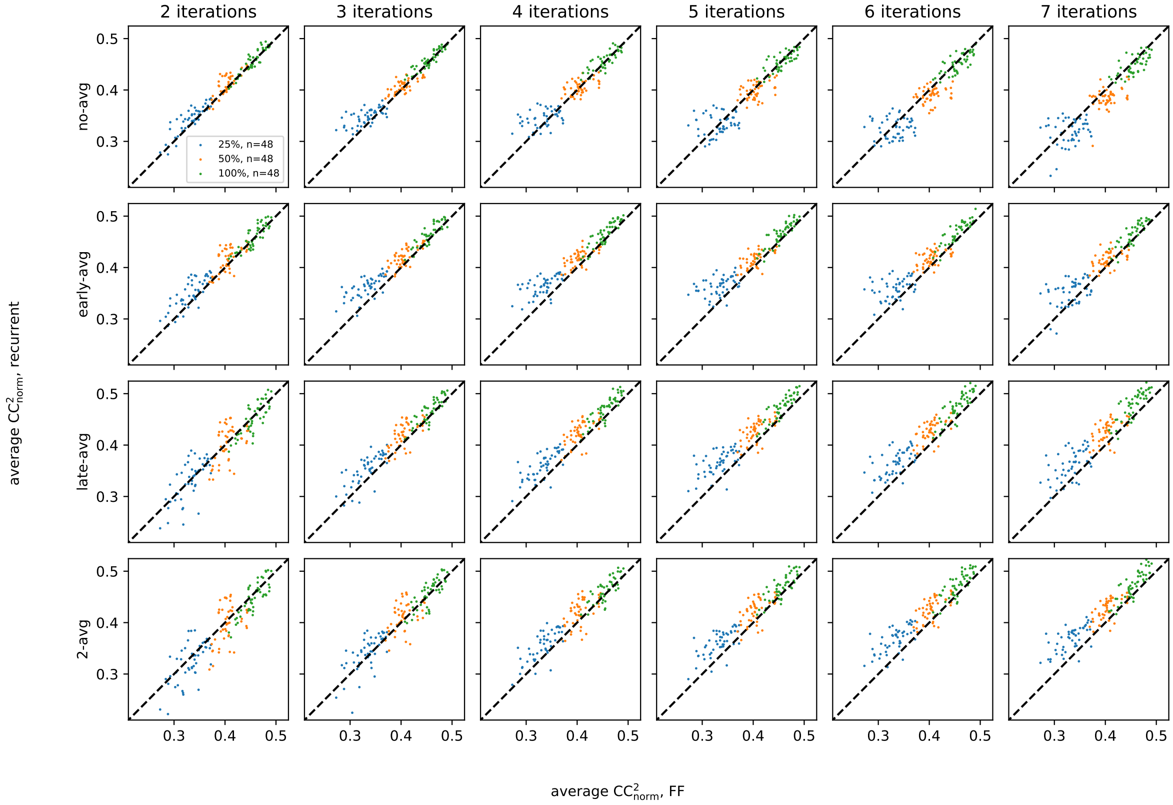
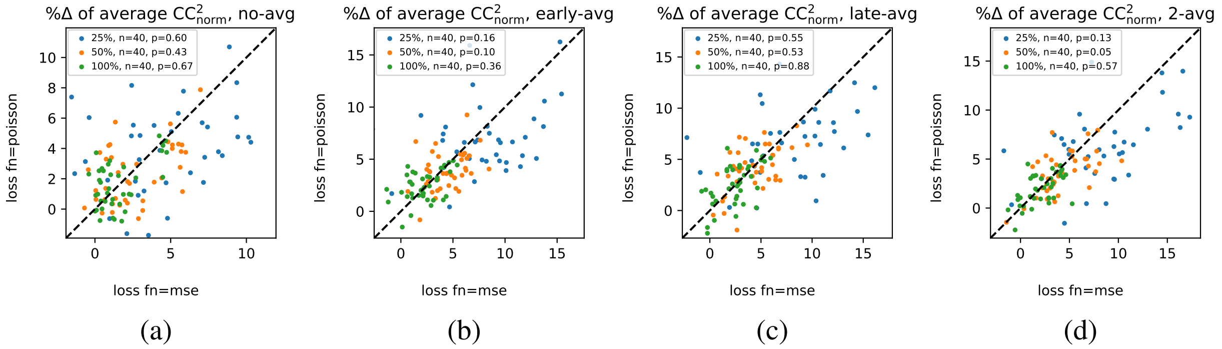


Figure 4.7: Recurrence models vs. similarly-sized feed-forward models under 24 combinations of readout mode and number of iterations, for NS 2250 data. The style of the figure follows that of Figure 4.6.

4.4.1.2 Recurrent models outperformed feed-forward models more under larger model size and less training data

As described in Section 4.3.3.2, we trained a huge number of models with different model size-independent hyperparameters and loss functions, to understand whether the advantage of recurrent models over feed-forward models is dependent on these details in model design. To quantify the advantage of the recurrent models over a feed-forward one, we computed the relative performance change of the best-performing recurrent model over the feed-forward one, for each one of the explored four readout modes.

For every model size-independent hyperparameter and loss function listed in Section 4.3.3.2, we tested whether the advantage of recurrent models over feed-forward models as defined above was dependent on the choice of model size-independent hyperparameters or loss functions by performing paired t-tests for all combinations of readout modes and amounts of training data. We did not find any model size-independent hyperparameter or loss function that consistently affected the advantage of recurrent models over feed-forward models, with almost all p values in tests larger than 0.05 (Figure 4.8, as well as Figures B.1,B.2,B.3,B.4,B.5 in Appendix B.8.1).



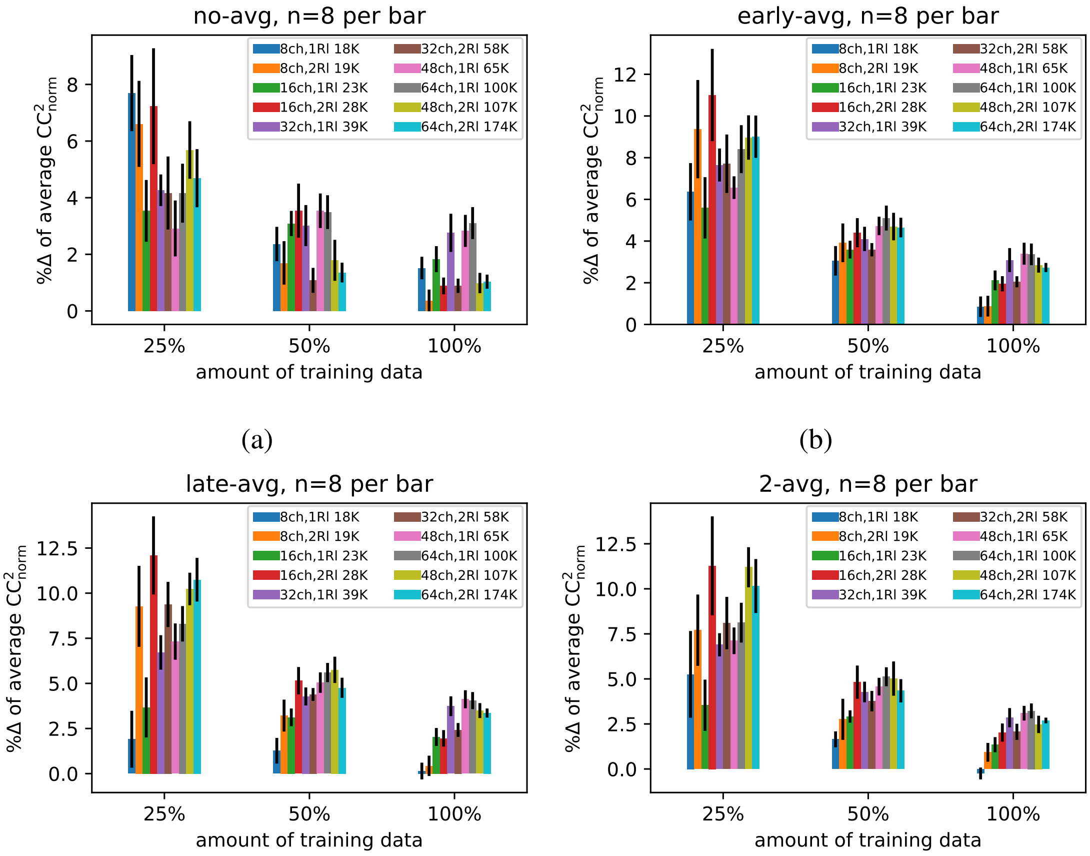
(a) (b) (c) (d)

Figure 4.8: The relative performance metric changes of recurrent models over feed-forward models under different loss functions on the ImageNet 8K data set. Results for the four different readout modes are shown in different panels. For each panel associated with a particular readout mode, we first averaged model performance metrics over model initialization seeds for all 480 feed-forward models and the corresponding 2880 recurrent models of this readout mode, resulting in 240 feed-forward model metrics and 1440 recurrent model metrics; we then computed the relative performance change of the best-performing (aggregated) recurrent model over the feed-forward one for every (aggregated) feed-forward model, resulting in 240 relative metric changes; afterwards we grouped the metric changes into 120 pairs with the same hyperparameters except the loss function as shown in the scatter plot, with models using mean squared loss on the horizontal axis and those using mean Poisson loss on the vertical axis; finally, we grouped the pairs by amount of training data and performed the standard two-tailed t-test for each explored training data size, with numbers of pairs n and p values shown in the legend of each panel. Results for other hyper parameters and data sets were similar and are included in Appendix B.8.1.

As we did not find the relative performance change of recurrent models over feed-forward ones to change much w.r.t. model size-independent hyperparameters or loss functions, we felt it was reasonable to aggregate model performance metrics over these hyperparameters and check whether the advantage of recurrent models over feed-forward ones could change with amount of training data and model size.

Figure 4.9 shows the relative performance change of recurrent models over feed-forward ones, under different amounts of training data and model sizes, on the ImageNet 8K data set. We found that generally speaking the recurrent model's performance advantage was larger when the model had more parameters or the amount of training data was less. The only outlier was the `no-avg` readout mode, which exhibited no clear relationship between model size and relative performance change. Similar results held on the NS 2250 data set (Figure 4.10).

The figures discussed above show relative performance gains of recurrent models over feed-forward ones; Appendix B.8.2 shows absolute performance metrics of models grouped by model size (Figures B.6,B.7,B.8 for the ImageNet 8K data set; Figures B.9,B.10,B.11 for the NS 2250 data set).



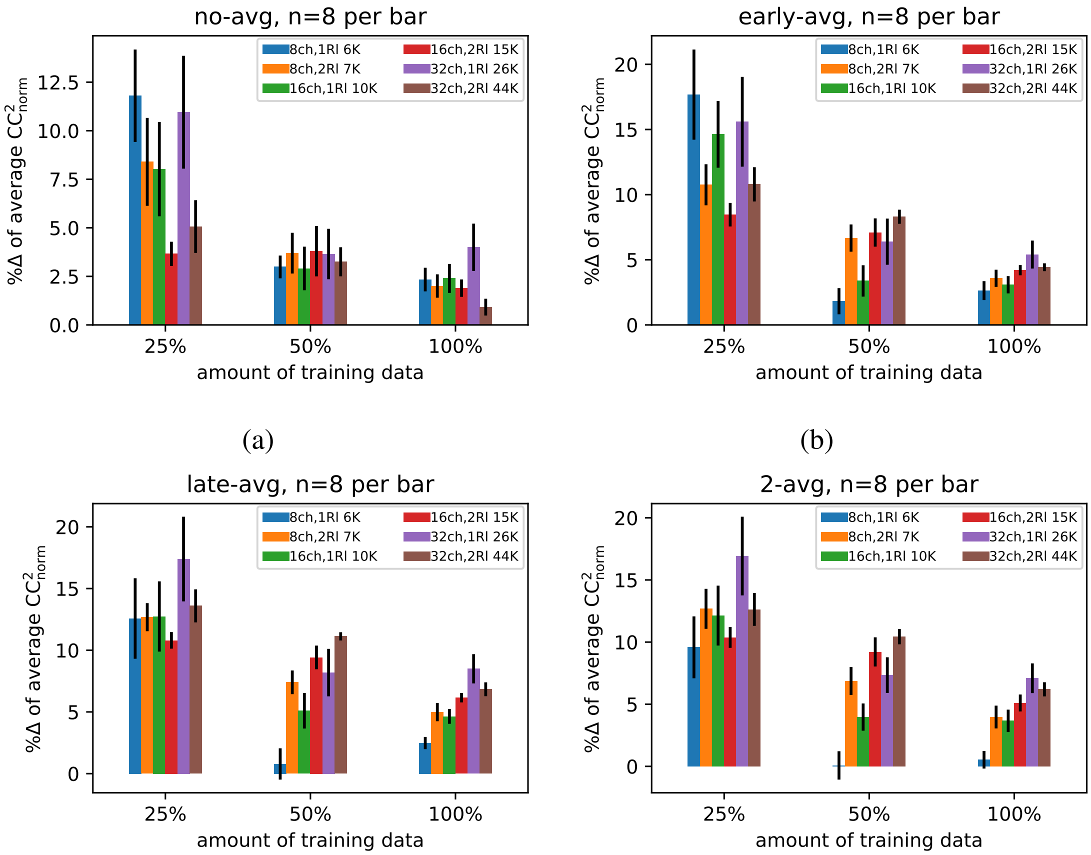
(a)

(b)

(c)

(d)

Figure 4.9: The relative performance changes of recurrent models over feed-forward models under different model sizes and amounts of training data on the ImageNet 8K data set. Each panel shows the results for recurrent models of a particular readout mode as indicated in the panel's title. For each readout mode, we first averaged model performance metrics over model initialization seeds for all 480 feed-forward models and the corresponding 2880 recurrent models of this readout mode, resulting in 240 feed-forward model metrics and 1440 recurrent model metrics; for each of the 240 (aggregated) feed-forward models, there were six (aggregated) recurrent models with matched hyperparameters and model size as the feed-forward one but different numbers of iterations; we computed the relative performance change of the best-performing (aggregated) recurrent model over the feed-forward one among the six, resulting in 240 relative metric changes in total; afterwards we aggregated the results over all three explored model size-independent hyperparameters and loss functions to obtain the the bar chart grouped by model size and amount of training data. Each bar's height denoting the average relative performance change under a particular combination of model size and amount of training data over $n = 8$ combinations of model size-independent hyperparameters and loss functions; error bars denote s.e.m. over model size-independent hyperparameters; legend shows the number of channels (`ch`), number of recurrent layers (`Rl`), and model size for every bar; under each amount of training data, bars are ordered by the model size.



(a)

(b)

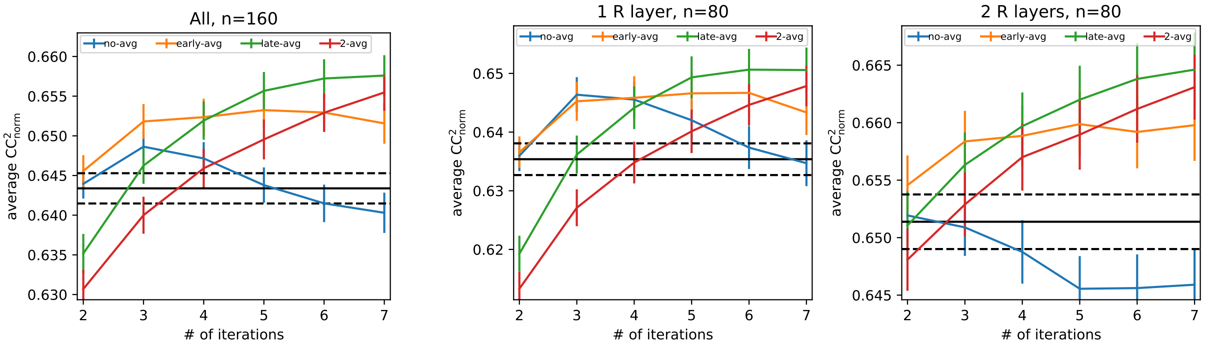
(c)

(d)

Figure 4.10: The relative performance changes of recurrent models over feed-forward models under different model sizes and amounts of training data on the NS 2250 data set.

4.4.1.3 Overall comparison of recurrent and feed-forward models

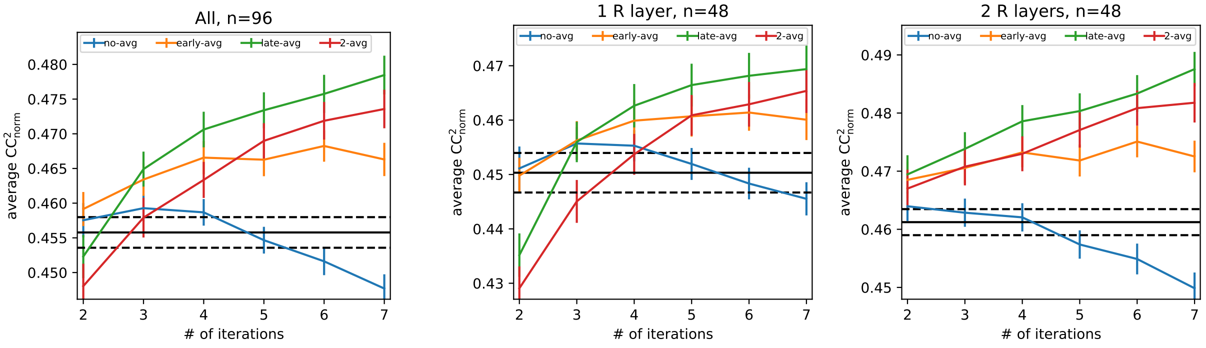
To get a summary overview of recurrent vs. feed-forward models, we further aggregated results over number of channels as well as number of layers. Figure 4.11 shows the recurrent models vs. feed-forward models on the ImageNet 8K data aggregated under different numbers of layers using all the available training data and Figure 4.12 show the corresponding results on the NS 2250 data. While we did not find the best-performing recurrent models' performance advantage over feed-forward ones to be dependent on the number of layers consistently (Figures B.12,B.13), we did find that 1-recurrent-layer models had some readout modes (`late-avg` and `2-avg`) perform worse than the feed-forward baseline when the number of iterations was small (1 or 2); in addition, models with `no-avg` readout mode tended to no offer much less performance gain compared to models with other readout modes. Similar results held with models trained under less training data as shown in Appendix B.8.3 (Figures B.14,B.15 for ImageNet 8k; Figures B.16,B.17 for NS 2250).



(a) (b)

Figure 4.11: Recurrent models vs. feed-forward models on the ImageNet 8K data. **(a)** shows the results averaged over all explored recurrent models trained using all available training data, corresponding to rows with 100% training data in Table 4.2; the two panels in **(b)** show the results averaged over models with one recurrent layer and two recurrent layers, respectively. In each panel, colored lines show the performance metrics of recurrent models with different numbers of iterations and readout modes; error bars show s.e.m. Black solid lines and dashed lines denote feed-forward models' metrics and s.e.m., respectively. n (160 for **(a)** and 80 for **(b)**) gives the number of models used to compute every line and error bar. All models shown here were trained using all available training data. The performance advantage of recurrent over feed-forward models held with other amounts of training data as shown in Appendix B.8.3.

Finally, we further aggregated model metrics over number of layers to get a summary of the performance gain of recurrent models as shown in in Table 4.2 and Figure 4.11a for the ImageNet 8K data; the summary for the NS 2250 data is shown in Table 4.3 and Figure 4.12a. With appropriate choices on readout mode and number of iterations, recurrent models outperformed feed-forward ones on the ImageNet 8K data set by around $6\,\%$ with $25\,\%$ training data and by around $2\,\%$ with all training data; on the NS 2250 data set, recurrent models outperformed feed-forward ones by around $10\,\%$ with $25\,\%$ training data and by around $4\,\%$ with all training data.



(a) (b)

Figure 4.12: The NS 2250 version of Figure 4.11. Data shown in **(a)** correspond to those rows with 100% training data in Table 4.3.

4.4.1.4 Summary

In summary, we have the following observations comparing recurrent models with feed-forward ones of matched size and hyperparameters.

- Recurrent models almost always outperformed feed-forward ones of matched size and hyperparameters, at the level of individual models (Section 4.4.1.1).
- We did not find much difference in the relative advantage of recurrent models over feed-forward ones regarding the choice of model size-independent hyperparameters or loss functions (Section 4.4.1.2).
- The relative advantage of recurrent models over feed-forward ones was consistent but generally became larger with larger model size and less training data (Sections 4.4.1.2). On average, recurrent models outperformed feed-forward ones by 2 % to 6 % on ImageNet 8K, and 4 % to 10 % on NS 2250, depending the amount of training data used (Section 4.4.1.3).
- When comparing recurrent models of different choices on number of layers, number of iterations, and readout modes, we found that 1) one-recurrent-layer models of `late-avg` or `2-avg` readout modes underperformed feed-forward ones when the number of iteration was small (one, two); 2) readout mode `no-avg` underperformed other ones. (Section 4.4.1.3).

Among all the hyperparameters, loss functions, and amounts of training data we explored (Section 4.3.3.2), we only found qualitative differences for the performance gain of recurrent models over feed-forward ones when we change the number of layers, number of iterations, or readout mode. Therefore, we will in the rest of this study mostly present results for models trained on 100% of training data, and group results by number of layers, number of iterations, and readout modes, with all other hyperparameters averaged out. Results for models trained for other amounts of training data can be found in Appendix B.8 and Appendix B.9 with similar conclusions.

Table 4.2: Recurrent-forward models vs. feed-forward models on ImageNet 8K data. Each model performance metric in the table is the average over 160 trained models with different hyperparameters. For each row, the performance metric of the best performing model over iterations $T = 2, \dots 7$ is shown in bold. The last column gives the relative performance difference of the best-performing (aggregated) recurrent model over the feed-forward one (column 2, FF).

training data	FF	readout	# of iter T=2	T=3	T=4	T=5	T=6	T=7	max % Δ
25% (1280)	0.4872	no-avg	**0.5021**	0.4961	0.4918	0.4812	0.4736	0.4669	3.0716
		early-avg	0.5107	**0.5153**	0.5137	0.5096	0.5056	0.5047	5.7691
		late-avg	0.4948	0.5092	0.5118	**0.5140**	0.5132	0.5111	5.5101
		2-avg	0.4887	0.5010	0.5099	0.5148	0.5158	**0.5189**	6.5039
50% (2560)	0.5896	no-avg	0.5949	**0.5974**	0.5929	0.5866	0.5803	0.5793	1.3237
		early-avg	0.6017	**0.6076**	0.6057	0.6036	0.6047	0.6042	3.0558
		late-avg	0.5907	0.6011	0.6057	0.6091	0.6101	**0.6104**	3.5247
		2-avg	0.5854	0.5975	0.6043	0.6052	0.6078	**0.6101**	3.4786
100% (5120)	0.6434	no-avg	0.6439	**0.6486**	0.6472	0.6438	0.6415	0.6403	0.8162
		early-avg	0.6456	0.6518	0.6524	**0.6532**	0.6529	0.6516	1.5324
		late-avg	0.6352	0.6462	0.6519	0.6557	0.6572	**0.6576**	2.2090
		2-avg	0.6307	0.6400	0.6459	0.6495	0.6529	**0.6555**	1.8787

Table 4.3: Recurrent models vs. feed-forward models on the NS 2250 data set. Each model performance metric in the table is the average over 96 trained models with different hyperparameters. It is the NS 2250 version of Table 4.2.

training data	FF	readout	# of iter T=2	T=3	T=4	T=5	T=6	T=7	max % Δ
25% (350)	0.3322	no-avg	0.3421	**0.3484**	0.3430	0.3329	0.3252	0.3186	4.8866
		early-avg	0.3491	0.3620	**0.3639**	0.3637	0.3593	0.3518	9.5596
		late-avg	0.3316	0.3526	0.3599	**0.3657**	0.3646	0.3619	10.0854
		2-avg	0.3230	0.3396	0.3542	0.3603	0.3666	**0.3685**	10.9439
50% (700)	0.4053	no-avg	**0.4116**	0.4083	0.4051	0.3985	0.3904	0.3842	1.5580
		early-avg	0.4147	**0.4195**	0.4192	0.4158	0.4177	0.4153	3.5044
		late-avg	0.4043	0.4204	0.4247	0.4271	**0.4288**	0.4274	5.8053
		2-avg	0.3978	0.4130	0.4198	0.4240	**0.4269**	0.4266	5.3230
100% (1400)	0.4558	no-avg	0.4575	**0.4593**	0.4587	0.4547	0.4516	0.4477	0.7682
		early-avg	0.4592	0.4634	0.4666	0.4663	**0.4682**	0.4663	2.7355
		late-avg	0.4523	0.4649	0.4706	0.4734	0.4758	**0.4785**	4.9779
		2-avg	0.4480	0.4579	0.4634	0.4690	0.4719	**0.4736**	3.9046

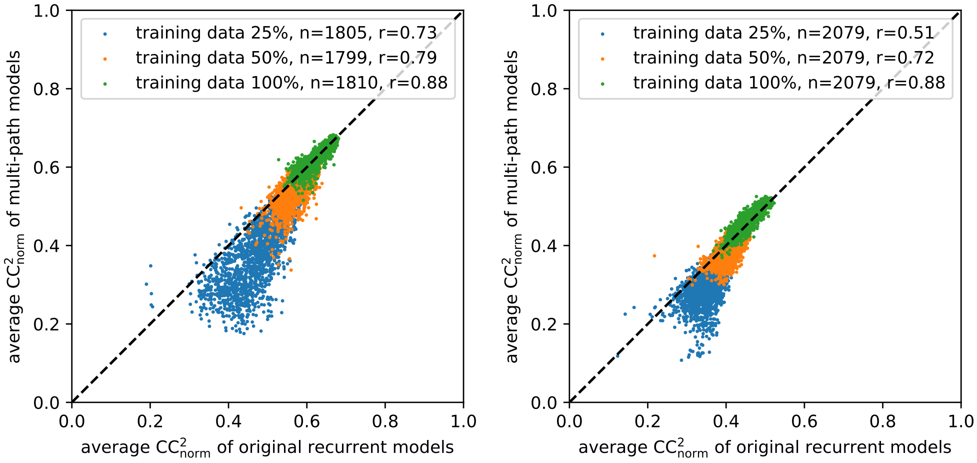
4.4.2 Approximation of recurrent blocks as multi-path ensembles

While recurrent models in general outperformed feed-forward models (Section 4.4.1), the reason behind the recurrent model's superiority was not immediately clear. In addition, the relationships among recurrent models trained with different readout modes and different numbers of layers were not clear either, as their performance metric curves differed a lot when the number iterations changed (Figures 4.11,4.12); in particular, one-recurrent-layer models of `late-avg` or `2-avg` readout modes underperformed feed-forward ones when the number of iteration was small and models of readout mode `no-avg` in general underperformed other ones (Section 4.4.1.4)

To better understand why recurrent models outperformed feed-forward ones, and to better understand the relationships among different recurrent models of different readout modes, we tried to reformulate each recurrent model as a multi-path model that is easier to analyze, as described in Section 4.3.4.

While the reformulation is approximate in nature as we ignored the effects of activation functions and bias parameters in the derivation, in practice we found that the reformulation largely held in the sense that the corresponding multi-path model trained from scratch and the original recurrent model were matched from multiple aspects. First, the retrained model and the original model matched in their performance metrics, especially when the amount of training data was large (Figure 4.13); second, the two matched in their multi-path ensembles' key characteristics, such as average path length as defined and used later in Section 4.4.3 to explain the differences among recurrent and feed-forward models (Figures 4.14,4.15). The correlation between recurrent and retrained multi-path models was lower when the amount of training data was small (results for 25% training data in Figures 4.13,4.14,4.15), most likely because retrained multi-path models had additional BN layers as described in Appendix B.5, which explains in detail the construction and training of multi-path models to approximate the original recurrent models.

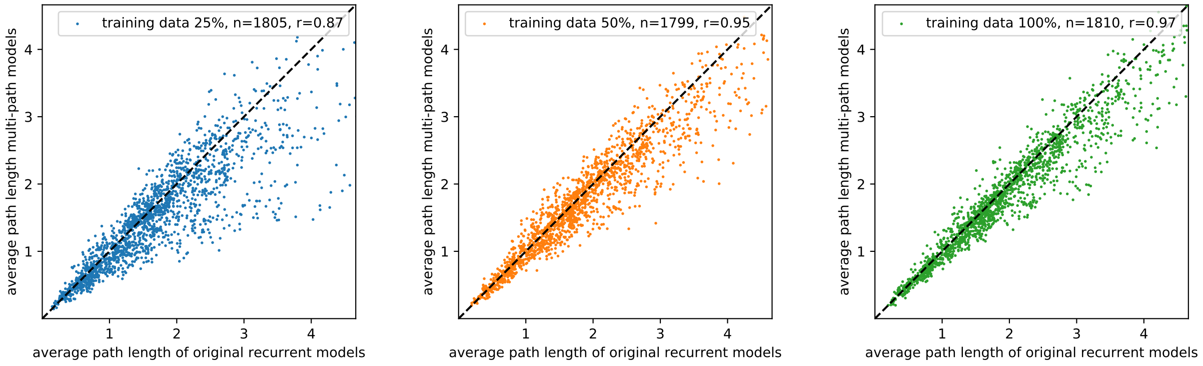
In short, through the above multi-path reformulation, we can convert each recurrent model into a corresponding model that contains a multi-path ensemble with shared parameters across individual feed-forward paths; the conversion allows us to compare all feed-forward and recurrent models we trained in a unified framework (feed-forward models can be thought as having a degenerate multi-path ensemble with a single path).



(a)

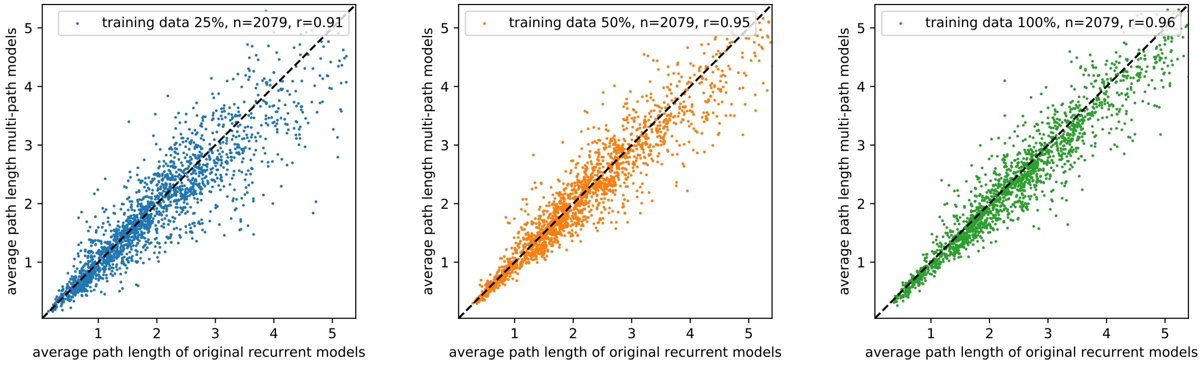
(b)

Figure 4.13: Performance metrics of multi-path vs. original recurrent models, with **(a)** showing results for ImageNet 8K and **(b)** for the NS 2250. For each dot, the horizontal axis indicates the performance of a recurrent model, which contains RCPB(s) with information flow shown in Figures 4.5a,4.5b, and vertical axis the performance of the corresponding retrained multi-path model where RCPBs are replaced by a multi-path ensemble as shown in Figures 4.5c,4.5d. See Appendix B.5 for details. To quantify the correlation of performance metrics between multi-path and original recurrent models, results are grouped according to the amount of training data; for each training data size, the number of model pairs n and the Pearson correlation r are shown in the legend.



(a) (b) (c)

Figure 4.14: Multi-path vs. original recurrent models in terms of their multi-path ensembles' average path lengths, on the ImageNet 8K data set. For a multi-path model containing a multi-path ensemble similar to those in Figures 4.5c,4.5d, the average path length can be computed following Section 4.4.3; for a recurrent model containing RPCB(s) similar to those in Figures 4.5a,4.5b, first its RPCB(s) are reformulated as a multi-path ensemble and then the average path length can be computed from the multi-path ensemble that contains the same trained parameters as the original RPCB(s), as shown in Figure 4.16. The three panels from left the right show results for models trained with 25%, 50%, and 100% of available training data, respectively. The correlations between models are shown in the legend, in the same style as that of Figure 4.13.



(a) (b) (c)

Figure 4.15: The NS 2250 version of Figure 4.14.

4.4.3 Understanding differences among models via path length and diversity

After establishing the approximate equivalence between recurrent models and models with multi-path ensembles, we then analyzed each original recurrent model's recurrent block(s) as a multi-path ensemble with shared parameters. Each multi-path ensemble has multiple paths of different lengths (Figures 4.5c,4.5d) and we found that differences among models could be mostly explained by the following two characteristics of the ensemble.

1. *Average path length*, weighted by the strength of each path. Conceptually the average path length can be considered the *effective depth* of an ensemble.
2. *Path diversity*. Models with similar average path lengths can be very different in terms of the contributions of individual paths. For example, an average path length of 3 can be either created by having a single path of length 3 or having five paths of lengths 1 through 5. Intuitively, the latter model has higher diversity than the former. In this study, we use the path length distribution as a measure of diversity.

Figure 4.16 demonstrates the computation of average path length and path diversity for an example recurrent model. Apart from the formulation of the recurrent information flow as a multi-path ensemble as described in Section 4.4.2, another key element essential to the computation of length and diversity is the concept of *strength score* for each component on a path. Roughly speaking, the strength score is a scalar that measures the ratio of the magnitude of the component's output over that of its input. In particular, we define the strength score of each component as follows.

1. The *strength score* of a BN layer is the average of the absolute values of the scaling factors used during inference, over all output channels.
2. The *strength score* of a convolutional layer is the average of the 2-norms of 3-D convolutional kernels flatten into vectors, over all output channels.
3. The *strength score* of an activation layer (ReLU or softplus in this study) is 1, as the activation layer outputs the input itself when the input is a large enough positive number.

The full definition of average path length and path diversity for a recurrent model is given in Appendix B.6.

In the following, we will compare the multi-path ensembles of recurrent and feed-forward models in terms of average path length and path diversity, to answer the questions posted in the beginning of Section 4.4.2 as reiterated below.

- In general, why did recurrent models outperform feed-forward ones?
- Comparing recurrent models trained with different readout modes and different numbers of layers, why did one-recurrent-layer models of readout modes `late-avg` and `2-avg` underperform feed-forward ones when the number of iteration was small?
- Comparing recurrent models trained with different readout modes and different numbers of layers, why did models with readout mode `no-avg` underperform other ones and what caused the differences among different readout modes in general?

Of the three questions listed above, the first two will be addressed in Section 4.4.3.1 and the third will be addressed in Section 4.4.3.2.

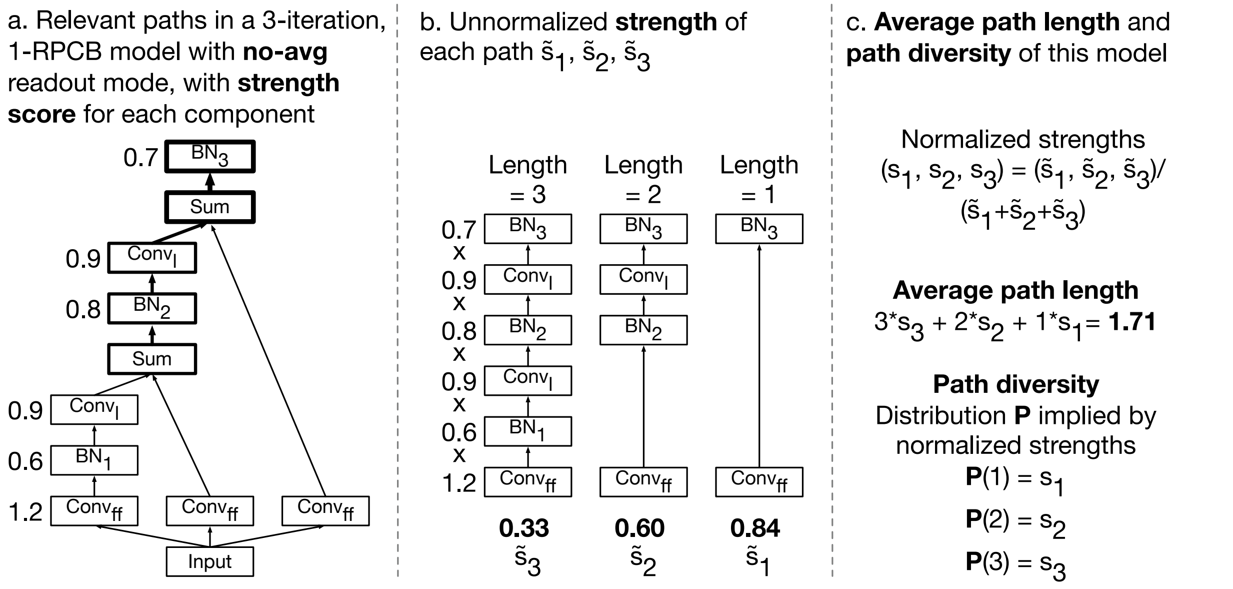

a. Relevant paths in a 3-iteration, 1-RPCB model with no-avg readout mode, with strength score for each component
0.7 BN3
Sum
0.9 Convl
0.8 BN2
Sum
0.9 Convl
0.6 BN1
1.2 Convff
Convff
Convff
Input
b. Unnormalized strength of each path s̃1, s̃2, s̃3
Length = 3
Length = 2
Length = 1
0.7 x 0.9 x 0.8 x 0.9 x 0.6 x 1.2
BN3
Convl
BN2
Convl
BN1
Convff
0.33 s̃3
0.60 s̃2
0.84 s̃1
c. Average path length and path diversity of this model
Normalized strengths
(s1, s2, s3) = (s̃1, s̃2, s̃3)/(s̃1+s̃2+s̃3)
Average path length
3*s3 + 2*s2 + 1*s1= 1.71
Path diversity
Distribution P implied by normalized strengths
P(1) = s1
P(2) = s2
P(3) = s3

Figure 4.16: Computation of average path length and path diversity for an example recurrent model. The model here has 3 iterations, one recurrent block (RCPB) and uses the `no-avg` readout mode. In **(a)**, the relevant information flow is shown, with a *strength score* assigned to each component. In **(b)**, the original information flow is reformulated as a simple multi-path ensemble with three paths, and the unnormalized strength of each path $\tilde{s}_1, \tilde{s}_2, \tilde{s}_3$ is computed as the product of strength scores along that path. In **(c)**, the *average path length* of the model is defined by the average path length weighted by the normalized strengths s_1, s_2, s_3, and the *path diversity* of the model is the probability distribution implied by normalized strengths s_1, s_2, s_3. The length of each path is defined as the number of convolutional layers in that path. Activation layers are omitted in **(a)** and **(b)** for brevity and they do not affect the calculation due to having a strength score of 1.

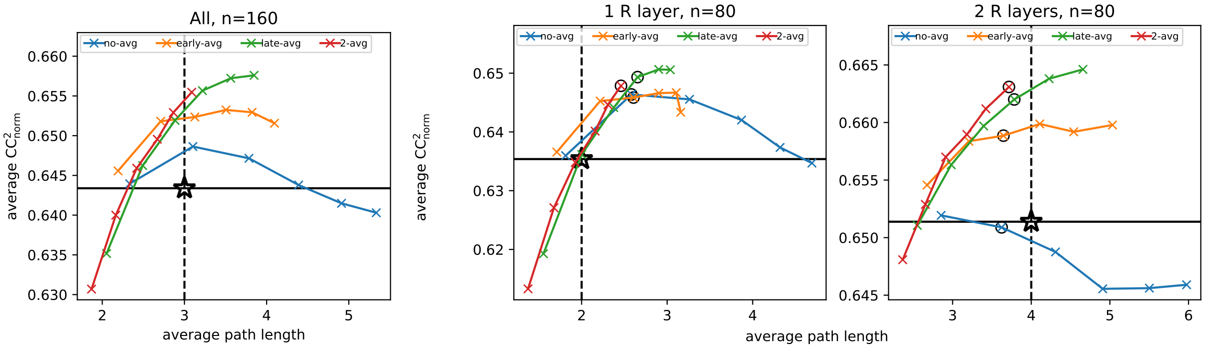
4.4.3.1 Recurrent models outperformed feed-forward ones of similar depths via implicit and compact multi-path ensembles

By comparing all models in terms of their multi-path ensembles, we can now easily see the that the performance differences between recurrent and feed-forward models could be attributed to the differences in their multi-path ensembles. Feed-forward models only contain a single path in their degenerate multi-path ensembles, whereas recurrent models contain diverse paths of different lengths in their implicitly implemented multi-path ensembles.

We hypothesize that the advantage of the recurrent model rests on the ensemble of multiple feed-forward paths embedded in the recurrent computation and such multitude of paths makes the recurrent model more flexible compared to a feed-forward model; a stronger and general version of our hypothesis is that recurrent models with more diverse sets of feed-forward paths outperform those with less diverse ones, with the least diverse recurrent model being a feed-forward model.

Figures 4.17,4.18 plot all models' performance vs. average path length, with recurrent models shown in lines and feed-forward models shown in black stars. We have the following observations consistent with our hypothesis.

1. When recurrent models have similar average path lengths as feed-forward ones (around dashed lines in the figures), recurrent models performed similarly as or better than feed-forward ones. The result was consistent with our hypothesis, because recurrent models achieved a similar average path using a more diverse set of paths compared to feed-forward models each of which only contains a single path.
2. The advantage of recurrent models relative to similar-path-length feed-forward models was larger when the average path length was around 4 (Figures 4.11b,4.12b, around dashed lines), compared to when the average path length was around 2 (Figures 4.11a,4.12a, around dashed lines). The result was consistent with the more general version of our hypothesis because the former set of recurrent models have more diverse sets of paths due to having more layers and more model iterations.
3. When recurrent models have higher average path lengths than feed-forward ones (right of dashed lines in the figures), recurrent models typically outperformed feed-forward ones more, probably because the multi-path ensembles were more diverse and complex.

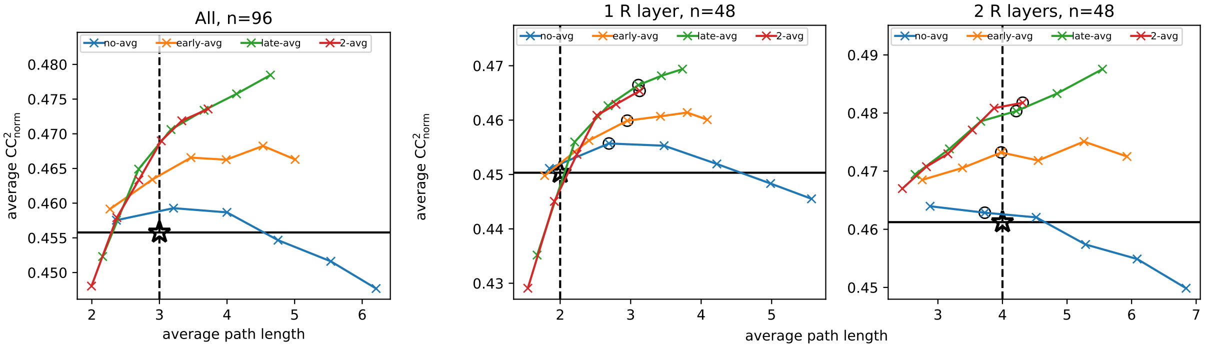


(a)

(b)

Figure 4.17: Performance vs. average path length for recurrent models on the ImageNet 8K data. It's the same as Figure 4.11 except that the horizontal axis is the calculated average path length instead of the number of iteration hyperparameter. In addition, error bars are omitted for clarity, and vertical dashed lines denote average path lengths of feed-forward models which are shown as stars. **(a)** shows the results averaged over all explored recurrent models trained using all available training data; the two panels in **(b)** show the results averaged over models with one recurrent layer and two recurrent layers, respectively. Black circles in **(b)** denote a set of models with similar average path lengths but different readout modes; they will be further analyzed in Figure 4.20a. Models here were trained on all available training data. Additional results for models trained with less data can be found in Appendix B.9.

Understandably, our hypothesis that recurrent models perform better due to diverse paths will not necessarily work when the recurrent model has a much lower average path length compared to the feed-forward one. For one-recurrent-layer models with `late-avg` or `2-avg` readout modes, models with lower average path lengths (left of dashed lines in the figures) performed

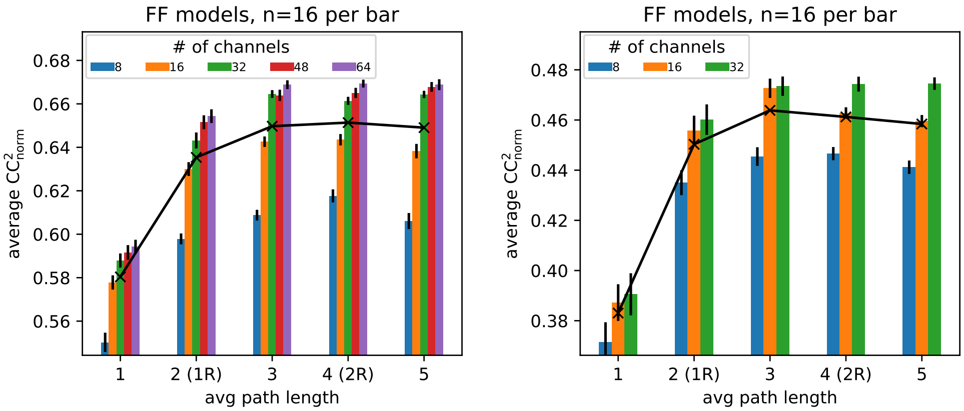


(a) (b)

Figure 4.18: NS 2250 version of Figure 4.17.

much worse than feed-forward models whereas we did not find similar issues for two-recurrent-layer models. By treating recurrent models as multi-path ones, we hypothesized the differences could be explained by the performance differences of feed-forward models of different path lengths. To verify our hypothesis, in addition to the 3- and 5-layer feed-forward models (or equivalently feed-forward models with average path lengths 2 and 4, respectively; we always compute average path length without counting the first convolutional layer) trained to match the model size with recurrent models, we trained feed-forward models with 2, 4, and 6 layers (or equivalently with average path lengths 1, 3, and 5). As shown in Figure 4.19, the performance metrics of feed-forward models increased a lot when the average path length changed from 1 to 2, but the increase became much smaller and incremental with higher average path lengths. Given the results of feed-forward models, we believe one-recurrent-layer models of lower average path lengths performed worse because lower-performing paths of length 1 contributed to much of these models' responses; two-recurrent-layer models of lower average path lengths performed similarly as size-matched feed-forward models because the multi-path ensembles of these models only contain paths longer than one and these paths all performed relatively similarly and much higher than a path of length 1.

The recurrent computation information flow in a recurrent model implicitly defines a compact multi-path ensemble with shared parameters across paths (Figure 4.5); we propose that such compact multi-path ensemble allows approximating the complex function underlying recurrent biological circuits with more flexibility compared to feed-forward model matched in model size and hyperparameters.



(a)

(b)

Figure 4.19: Performance metrics of feed-forward models with different average path lengths, trained on ImageNet 8K **(a)** and NS 2250 **(b)** using 100% of training data. In each panel, results are grouped by average path length and number of channels. Each cluster of bars along the horizontal axis show results for models with the same average path length and different bars in a cluster show results for models with the same number of layers and the same number of channels. The height of each bar denotes the model performance averaged over 16 models, and error bars denote s.e.m. In addition, the line plot on top of bars shows the average performance metrics for each explored number of channels, averaged over 80 models for ImageNet 8K **(a)** and 48 models for NS 2250 **(b)**. The depth of a feed-forward model is measured in its average path length, to match the horizontal axis of Figures 4.17,4.18. Alternatively, a feed-forward model's depth can be measured in number of layers. A feed-forward model with x layers has an average path length of $x-1$ because we always omit the initial convolutional layer for the calculation of average path length; a feed-forward model with 3 layers, or an average path length of 2, is of matched size with a recurrent model with one recurrent layer (`1R`); a feed-forward model with 5 layers, or an average path length of 4, is of matched size with a recurrent model with two recurrent layers (`2R`). Results for models trained using less training data are given in Appendix B.9.

4.4.3.2 Differences in path length and diversity could explain performance differences across readout modes

While the previous section demonstrates that recurrent models outperformed feed-forward ones due to the former's diverse multi-path ensembles, the differences among different readout modes at different numbers of iterations are still be to accounted for. In Figures 4.11,4.12, performance vs. number of iteration curves for different readout modes were not well aligned, crossing each other at different vertical axis locations (especially Figures 4.11b,4.12b). In other words, no obvious connections and similarities among readout modes could be found from the curves. By reformulating recurrent models as multi-path models, (Section 4.4.2), we found that performance differences among readout modes and iterations were highly correlated with differences in the path length and diversity of the corresponding multi-path ensembles.

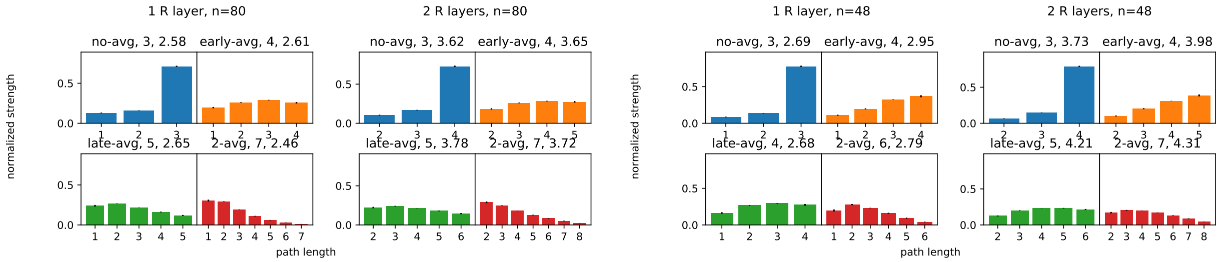
4.4.3.2.1 Path length aligned different readout modes better than number of iteration As shown in Figures 4.17,4.18, the curves of performance vs. average path length were much better aligned across different readout types, compared to Figures 4.11,4.12. In particular, when the average path length was relatively short (say, ≤ 2.5), all readout modes' curves were roughly the same; and when the path length was higher, curves began shooting in different directions depending on the mode, due to differences in the path diversity, as explained in the following.

4.4.3.2.2 Shorter paths played more important roles than longer ones While average path length could help unify different recurrent readout modes when the length was short, readout modes did behave differently as the length increased. In general, under similar average path lengths, `late-avg`/`2-avg` modes performed the best, the `early-avg` performed slightly worse, and the `no-avg` performed much worse. (Figures 4.17,4.18).

Just as recurrent models outperformed feed-forward ones due to having more diverse multi-path ensembles (Section 4.4.3.1), we hypothesized that performance differences between readout modes were related to path diversity, when the average path length was similar. Figure 4.20 shows the path length distributions for different readout modes when their average path lengths were similar as denoted by black circles in Figures 4.17b,4.18b.

Under similar average path lengths, different readout performed differently, with different path length distributions. In general, as the shorter paths contributed relatively more in the overall distribution, the performance increased. The `no-avg` readout mode performed the worse, with the path length distribution dominated by longest paths (blue distributions in Figure 4.20); the `early-avg` readout mode performed better, with shorter paths contributing more to the distribution (orange distributions in Figure 4.20); the `late-avg` and `2-avg` modes performed the best, with even more contributions from shorter paths (green and red distributions in Figure 4.20).

With the above observations, we hypothesized that shorter paths contributed more to the model performance compared to longer ones. To verify the hypothesis, we performed ablation studies on 7-iteration models by removing paths of certain lengths from the full models that have all paths. Consistent with our hypothesis, models with shorter paths ablated outperformed those with longer paths ablated, across readout modes (Figures 4.21,4.22): when three paths were removed (right panel in each sub figure), removing shorter paths (1 to 3, 2 to 4, highlighted in black circles) gave larger performance decrease compared to removing longer paths which sometimes even gave performance boost. Note that there might be some interchangeability between paths of adjacent lengths, as removing a single path (left panel in each sub figure) had little impact on performance.



(a)

(b)

Figure 4.20: Path length distribution of example models with similar average path lengths across different readout modes, for models trained on ImageNet 8K **(a)** and NS 2250 **(b)**. For each data set, results for models with one recurrent layer are shown on the left, and those for models with two recurrent layers are shown on the right. For each data set and number of recurrent layer, results for the four different readout modes are shown in four smaller panels, with different readout modes shown in bars of different colors. In each of these smaller panels for a specific readout mode with a specific number of recurrent layer and a certain data set, the bar plot shows the path length distribution as defined in Section B.9, averaged over n models, with error bars showing s.e.m. The title given on top of each of these smaller panels show the readout mode's name, the number of iterations of the examined as circled in Figures 4.17,4.18, and the average path length of the (aggregated) model at that number of iterations. Models here were trained on all available training data. Additional results for models trained with less data can be found in Appendix B.9.

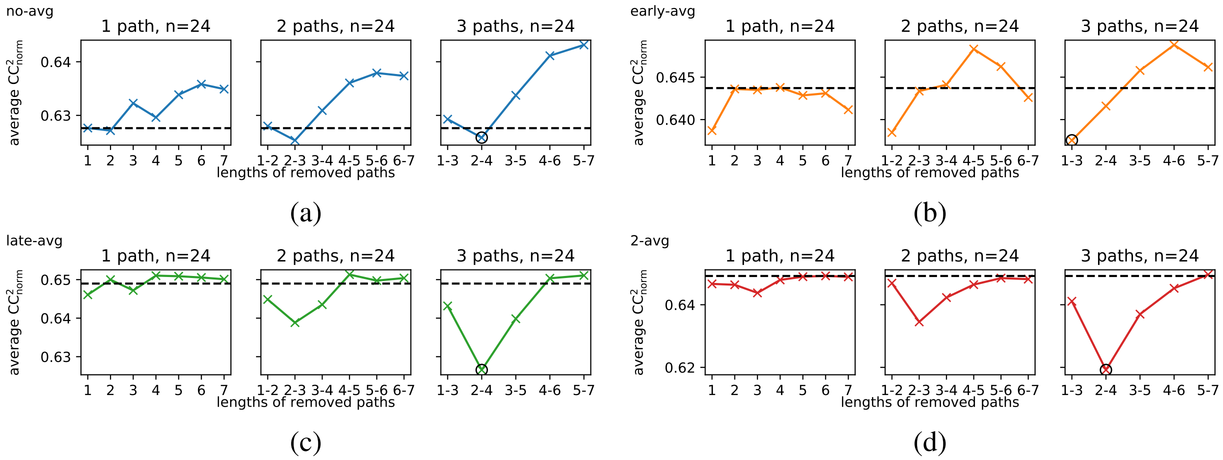


Figure 4.21: Performance metrics of ablated 2-layer, 7-iteration models of 16 and 32 channels on ImageNet 8K data, with 1, 2, or 3 paths removed at a time. Different panels **(a)** through **(d)** show results under different readout modes as denoted on the top left corner of each panel, and each panels shows results of models with 1, 2, 3 paths removed from left to right. Dotted lines show the performance metrics of full models that have all paths. Each circle denotes the worst performing ablation configuration with 3 paths removed. For each readout mode, on top of each of the four sub panels, the n in the title denotes the number of underlying models used to compute each data point. All models shown here were trained using all the available training data as the multi-path ensemble framework worked better with more data (Section 4.4.2). Models with 3-layers or more channels were not studied due to GPU memory limits as explained in Appendix B.7.

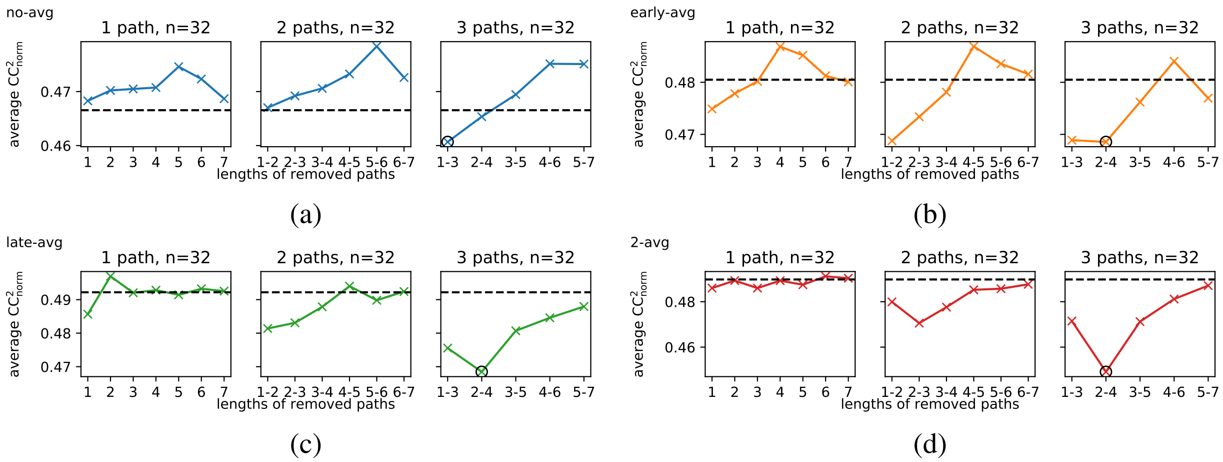


Figure 4.22: NS 2250 version of Figure 4.21.

4.5 Discussion

4.5.1 Key contributions of the study

By training tens of thousands of models on multiple neural data sets and employing a novel method that reformulates recurrent models as multi-path ensemble models, we have made the following contributions to the research of early visual areas and recurrent models.

Recurrent models could explain neural responses of early visual areas better than typical feed-forward models with matched hyperparameters and model sizes, especially when training data were less and the model size was larger. We show the relevant experimental results in detail in Section 4.4.1 with a comprehensive comparison between recurrent models and feed-forward models trained under tens of thousands of settings, with different data sets, different amounts of training data, and different model hyperparameters (Section 4.3.3.2). Overall, recurrent models outperformed feed-forward ones of matched size and hyperparameters almost consistently regardless of the setting, with some small differences for models with different numbers of layers and different readout modes (Section 4.4.1.4). While the performance advantage of recurrent models over feed-forward ones was consistent, the magnitude of the advantage was larger under larger model size and less training data (Section 4.4.1.2); in other words, recurrent models showed more advantage over feed-forward ones when the amount of training data per model parameter was less. While it will be certainly too speculative to claim that the architecture of our recurrent models can be mapped to neural circuitry of the brain faithfully, it's likely that our recurrent models captured some of the fundamental computational principles of the visual areas as a model prior that helps neural data prediction under less data.

The performance advantage of the recurrent model over the feed-forward one can be attributed to the former's compact and implicit multi-path ensemble that allows approximating the complex function underlying recurrent biological circuits with efficiency. We established the observation in two steps. First, we reformulated recurrent models as models with multiple feed-forward paths with shared parameters (Section 4.4.2); thus we were able to compare recurrent models and feed-forward models in a unified framework, with feed-forward models being degenerate multi-path models. Second, we compared feed-forward models and recurrent models using two key characteristics of their multi-path ensembles—average path length and path diversity (Section 4.4.3)—and found there was arguably a positive correlation between the path diversity of recurrent models and their performance advantage over feed-forward models (Section 4.4.3.1). The recurrent computation information flow in a recurrent model implicitly defines a compact multi-path ensemble with shared parameters across paths (Figure 4.5); we propose that such compact multi-path ensemble allows approximating the complex function underlying recurrent biological circuits with more flexibility compared to feed-forward model matched in model size and hyperparameters. The usefulness of multi-path ensemble in neural data prediction could be due to the inherent advantage of ensemble models for machine learning [7], or due to some similarity between multi-path architectures and neural circuits, or due to a bit of both.

A balance of short and long paths in the ensemble is necessary for the recurrent model to achieve the best performance; on our data sets, models with more relative weights on shorter paths tended to perform better than models with more relative weights on longer paths. We established the observations by comparing different readout modes (Section 4.3.1.2.2) in terms of their multi-path ensembles. In particular, we found that better performing readout modes have their path length distribution more concentrated at shorter paths (Figure 4.20) and removing shorter paths had more impact on the performance than removing longer paths in an ablation study (Figures 4.21,4.22). The relative importance of shorter paths over longer ones might be data set- and task-dependent and requires further investigation. For recurrent models, different readout modes essentially represent different weight assignments to intermediate outputs at different iterations (Section 4.3.1.2.2); our exploration of different readout modes show that the "vanilla" weight assignment, or `no-avg` readout mode, performed much worse compared to other readout modes; the comparison of different readout modes in terms of performance metrics and multi-path ensembles shows that a balance of early and late information across iterations, or equivalently a balance of short and paths in the corresponding multi-path ensemble, is necessary for the recurrent model to achieve the best performance.

The novel method that reformulates recurrent models as multi-path ensemble models provides a new perspective to understand recurrent models and connect recurrent models to the existing literature on ensemble models in the machine learning community. The key idea behind the reformulation method is that the complex information flow in recurrent computation can be approximated as a summation of multiple feed-forward paths with shared parameters (Sections 4.3.4). While the reformulation method is approximate by nature, the approximation was accurate enough in practice (Section 4.4.2) and we successfully explained the differences between recurrent models and feed-forward models as well as the differences among recurrent models by comparing the models' multi-path ensembles. It will be interesting to check the accuracy of our reformulation method in other machine learning tasks involving recurrent models or multi-path models with shared information flow among paths; understanding these complex models as simpler models summing over multiple paths can enable alternative model dissection and ablation methods for better understanding the merits of those complex models. For example, in many architecture search studies for recurrent networks [22, 47], authors mostly try to justify some architectures over others using performance metrics alone; it will be interesting to apply our reformulation methods to those different architectures and see if different architectures also yield different multi-path ensembles. In addition, our reformulation method and analysis techniques can be applied to those recurrent models performing well in computer vision tasks [48, 93, 116]; these studies typically have little or no explanation on the performance advantage of recurrent models from a computational perspective and the authors often justify recurrent models by the abundance of recurrent connections in the brain [32]; our reformulation method might reveal additional insights on the performance advantage of recurrent models.

4.5.2 Differences and relationships with relevant studies

4.5.2.1 Modeling temporal dynamics versus mean firing rates

Most studies on visual areas involving recurrent models have tried to model temporal dynamics of neural data [77, 93, 116, 155], whereas our study trained recurrent models to predict average firing rates instead of temporal dynamics of visual neurons. We chose to model average firing rates because we wanted to know in addition to modeling temporal dynamics, whether there is additional benefit of using recurrent models for neural data modeling; after all, by construction feed-forward models are bound to perform worse for modeling temporal dynamics. Our study demonstrated the superiority of recurrent models in a more competitive setting where feed-forward models are more likely to perform well compared to modeling temporal dynamics. Our results on neural prediction are conceptually consistent with many computer vision-oriented studies [48, 93, 115, 116] who have demonstrated that recurrent models can outperform feed-forward ones in static image classification tasks which do not involve explicit temporal dynamics. In many of these studies as well as ours, the internal dynamics of these models trained on tasks without explicit temporal dynamics could be useful for modeling neurophysiological data or effects involving temporal dynamics; in Kubilius et al. [93], Nayebi et al. [116], the internal model dynamics have been reported to resemble actual monkey IT population dynamics; in our study (to be described later in Section 4.5.2.2), the internal model dynamics exhibited longitudinal facilitation and lateral suppression of oriented bars [72] as well as reduction in orientation bandwidths and spatial frequency bandwidths over time [9, 137]. The above results of using internal dynamics learned from static tasks to model temporal dynamics of neurophysiological data and effects suggest that there might be some general benefit about recurrent modeling regardless of tasks.

4.5.2.2 Existing work on modeling early visual areas

Neural network-like models have long been used to model neural responses of early visual areas. Before the wide adoption of deep learning techniques, David and Gallant [25] tried to predict V1 neural responses using a fully connected layer followed by ReLU activation layer; Prenger et al. [132] modeled V1 neural data using more traditional multi-layer fully connected neural networks with sigmoid activation layers; Lau et al. [95] used neural networks to model and understand complex cells of V1; generalized linear models (GLMs) [75, 128], subunit models [172], nested GLMs [112], sparse deep belief nets for V2 [96], and boosting-based linear combinations of V1 models [176] can all be formulated as neural networks with specific network architecture designs and/or optimization techniques. In recent years, convolutional neural networks (CNNs) and its extensions have achieved great success for modeling early visual areas [13, 29, 81, 184]. Our study extends the above CNN work by applying recurrent CNNs to neural modeling tasks; more important, our novel reformulation method based on multi-path ensembles provides insights into the performance advantage of recurrent models.

While our study is one of the earliest ones to introduce lateral recurrent connections into CNNs for predicting mean firing rates of neurons in early visual areas, alternative mechanisms can introduce lateral connections to the model. Notable alternative mechanisms include divisive normalization [15] and dynamical system implementations of sparse coding [79, 140, 185]. All of these mechanisms might be implementing similar underlying computational principles of the brain, as all these models are able to reproduce many neurophysiological phenomena associated with recurrent circuits. Preliminary results in Rockwell et al. [138] showed that the internal model dynamics of our recurrent models exhibited longitudinal facilitation and lateral suppression of oriented bars [72] as well as reduction in orientation bandwidths and spatial frequency bandwidths over time [9, 137]; models incorporating divisive normalization and sparse coding mechanisms are able to explain many non-classical receptive field effects of V1 neurons as well [15, 185]; in addition, divisive normalization mechanims can be incorporated into CNNs, with improved model interpretability and little loss in performance [12].

4.5.2.3 Existing work connecting recurrent models and ensemble models

There have been relatively few studies connecting recurrent models to multi-path ensemble models in the deep learning context. For ResNets, Liao and Poggio [104] have pointed out the equivalence between a specific type of recurrent networks with weight-tied ResNets, and Chen et al. [16] formulate deep weight-tied ResNets as approximating the dynamics of continuous recurrent models specified in ordinary differential equations. To the best of our knowledge, there is no work explicitly connecting recurrent models to multi-path ensemble models in general. The most similar work to ours is probably Zhang et al. [181], which characterizes recurrent models in terms of some architectural complexity measures (feed-forward depth, recurrent depth, and recurrent skip coeffcient) motivated by graph theory and quantified the relationship between performance and these complexity measures. While conceptually similar, our work differs from Zhang et al. [181] in two important aspects. First, complexity measures in Zhang et al. [181] are entirely based on model architecture hyperparameters, whereas our measures of the recurrent model's multi-path ensemble depend on both model hyperparameters as well as learned model parameters; second, Zhang et al. [181] have developed their measures in the context of sequence modeling where both inputs and outputs of the task are variable-sized sequences, whereas we developed our measures of the multi-path ensemble for tasks with fixed-sized input and output.

4.5.2.4 Alternative methods to deeply understand recurrent networks

In the deep learning community, recurrent networks have often been analyzed in a less rigorous fashion. For example, the practical performance advantage of gated recurrent networks, such as LSTM [58] and GRU [17], over vanilla RNNs have often been attributed to the gate units' ability to alleviate gradient vanishing [46]; in addition, various kinds of recurrent model hyperparameter grid search have been performed for recurrent networks to find and justify the best architectures for certain tasks [22, 47]. Certain types of recurrent models have been deeply analyzed in alternative ways. Hinton et al. [57] have proved that an infinite logistic belief net with tied weights is equivalent to a restricted Boltzmann machine [55]; in a more relevant study, Bai et al. [5] have found that modern weight-tied deep recurrent networks in NLP tasks converge to a fixed point of the recurrent dynamics.

While our approach to understand recurrent models as multi-path ensembles succeeded in explaining the differences among recurrent and feed-forward models (Section 4.4.3), we also explored whether the empirical observations about the convergence of recurrent models made in Bai et al. [5] were applicable to our models. It turned out we failed to find any convergence to a fixed point for the internal dynamics of our models. The failure might be due to two reasons. First, the empirical observations about convergence in Bai et al. [5] probably only apply when the model dynamics is strictly recurrent, whereas our models actually follows a semi-recurrent dynamics with shared convolutional layers and independent normalization layers (Sections 4.5.3.1,B.2.2). Second, following the practice in the neural modeling literature [13, 81], we used activation layers without bound in the output (ReLU, softplus) whereas the models explored in Bai et al. [5] mostly use activation layers with bounded output (tanh, sigmoid); we tried training some of our models using tanh or sigmoid activation layers and obtained much worse results in terms of neural prediction metrics.

4.5.2.5 Alternative mechanisms of efficiently increasing the receptive field size

From a computational perspective, among other things, our recurrent models increase the effective receptive field size of output units via lateral recurrent connections. In the machine learning community, alternative approaches such as non-local neural networks [174] and self-attention [166] can also effectively increase the receptive field size of output units with efficiency in model size. It will be interesting to compare recurrent networks with these alternative approaches in the neural data modeling domain.

4.5.3 Limitations and future work

4.5.3.1 True recurrent dynamics versus semi-recurrent dynamics with shared convolutional layers and independent normalization layers

As noted in Sections 4.3.1.2,B.2.2, For our recurrent models, batch normalization is applied independently across time, following Spoerer et al. [155]. While this means that our models are not truly recurrent due to the independent normalization layer at each time step, our main conclusion that (semi-) recurrent models outperformed feed-forward models of matched size and hyperparameters still held (Section 4.4.1), as independent batch normalization layers add less than 1% to the overall model size, whereas increasing the number of channel or any other hyparparameter (kernel size, number of layers, etc.) by one generally induces an increase of model size of greater than or equal to 5%; in other words, model size increase cannot explain the performance advantage of our recurrent models over feed-forward ones. In addition, many other studies [48, 93, 155] that apply recurrent models to neural modeling or computer vision tasks have used independent normalization layers across time as well, potentially in order to address stability issues in model optimization.

4.5.3.2 Relative versus absolute performance gain of recurrent models

While recurrent models consistently outperformed feed-forward models matched in model size and hyperparameters, the absolute performance increase was still modest, compared to the amount of variance yet to be explained in the neural data. Adding feedback recurrent connections to our models that only contain lateral recurrent connections, exploring alternative readout modes (Section 4.3.1.2.2), and exploring other basic network components rather than convolution might help to fully capture the neural computation of early visual areas.

4.5.3.3 Connection of our work to the brain

We did not try to map the recurrent circuits in our best-performing models to the biological recurrent circuits in the brain. In general, given the nonconvex nature of neural networks, all recurrent neural network architectures in general might be equivalent in terms of performance given sufficient hyparparameter tuning [22]. However, it's likely that recurrent circuits benefit the brain in the same way as they benefit our models; in particular, biological recurrent circuits form compact and implicit multi-path ensembles that allow the brain to learn complex neural responses to visual stimuli with data efficiency. Given that we demonstrated the advantage of recurrent models consistently in two data sets with different stimulus presentation paradigms (Section 4.3.2) in modeling mean firing rates instead of temporal dynamics (Section 4.5.2.1), it's likely that our recurrent circuits capture some general aspects about neural computation in the early visual areas rather than any specific circuits of the brain.

To better map the recurrent circuits in the model to those in the brain, additional metrics other than neural data prediction under natural stimuli can be used for model comparison. For example, Kubilius et al. [93] leveraged existing knowledge about brain anatomy to constrain the number of layers in their recurrent models, and Rajaei et al. [133], Spoerer et al. [154] justified the validity their recurrent models by comparing model prediction and neural data under occluded object recognition tasks; in addition, neurophysiological experiments can be performed on models to see if they can produce various neural response properties (surround suppression, end-stopping, etc.) thought to be related to recurrent circuits [185]. Preliminary results in Rockwell et al. [138] showed that the internal model dynamics of our recurrent models exhibited longitudinal facilitation and lateral suppression of oriented bars [72] as well as reduction in orientation bandwidths and spatial frequency bandwidths over time [9, 137].

Regarding the connection between the brain and neural network-based methods as a whole, there have been numerous studies comparing CNNs with biological data at the neural level, at the behavior level, and from many other aspects [105]; while there are still many inconsistencies between CNNs and the biogloical visual system, generally speaking for now CNNs can be considered a viable model for the brain.

4.5.3.4 Testing the reformulation method in other machine learning contexts

Our novel reformulation method provides a new perspective to understand recurrent models and connect recurrent models to the existing literature on ensemble models. As discussed in the end of Section 4.5.1, applying our reformulation method and analysis techniques to other recurrent model-related studies, such as architecture search studies [22, 47] for recurrent models as well as computer vision studies [48, 93, 116] involving recurrent models, might yield additional insights on the advantage of recurrent computation for machine learning tasks in general.

Appendix A

Supplementary information for the second study

A.1 Results

A.1.1 More results on CNNs vs. other models

Figure A.1 shows how model performance changed with amount of training data. CNN models outperformed GLMs all the time. Figure A.2 shows the performance of CNN models with different numbers of parameters. Figures A.3 and A.4 show additional results on transfer learning using VGG networks. VGG19 overall worked similarly to or better than other ones.

A.1.2 Pilot experiments for selecting CNN architecture and optimization parameters

We performed the following pilot experiments to determine the architecture and optimization hyperparameters of our CNN models.

First, we made an relatively exhaustive list of candidate one-convolutional-layer architectures and candidate optimization hyperparameters for them. We focused on CNNs with only one convolutional layer as they are easier to analyze and also easier to compare with other models (Gabor models and GLMs, Section 3.3.4 and Table 3.1).

We tried two kernel sizes for convolutional layer hyperparmeters: 9 and 13. For each convolutional layer kernel size, we tried two basic readout layer architectures: pooling following by a vanilla fully connected layer, or a factored readout layer without pooling as in [13, 81]. For kernel size 9, we tried four pooling strategies `k8s4`, `k6s6`, `k6s2`, and `k3s3`, where `k` and `s` denote kernel size and stride of the pooling layer respectively. For kernel size 13, we tried three pooling strategies `k4s4`, `k6s2`, and `k2s2`. In addition, when a vanilla fully connected layer was used, we tried two versions: one with dropout and one without.

For optimization hyperparameters, we optimized them over four aspects: weight decay on convolutional layer, weight decay on readout layer, base optimizer, and learning rate. For (L2) weight decay on convolutional layer, we tried two configurations: 0.001 and 0.0001. For weight decay on readout layer, we tried four configurations: 0.001 L1, 0.001 L2, 0.0001 L1, and 0.0001 L2. For base optimizer, we tried Adam and vanilla SGD with momentum. For learning rate, we tried 0.001, 0.002, 0.005 for Adam, and 0.1 for SGD. We optimized the set of optimization hyperparameters over all combinations of candidate configurations of the these four aspects.

Then we evaluated the performance of each combination of these architectures and sets of optimization hyperparameters on 14 neurons chosen from monkey A based on CC_{max} (Section 3.3.6); in particular, from each of the seven neuron subclasses (Section 3.2.2.0.1), we picked the top two neurons in terms of CC_{max}, whose high value indicates high recording stability. We have the following empirical observations about different architecture hyperparameters and optimization hyperparameters for their performance.

- For any given model architecture, the best model performance over the four sets of optimization hyperparameters in Table 3.3 was very close to the best model performance over all sets of optimization hyperparameters explored. Therefore, we chose to use those four in our other experiments.
- Architectures with kernel size 9 in the convolutional layer overall outperformed those with kernel size 13. Among those with kernel size 9 in the convolutional layer, those with a pooling layer of configuration `k6s2` worked better than other schemes (other pooling configurations or factorized readout layer). Accordingly, we chose the baseline CNN architecture (Figure 3.2). Note that the main motivation of pooling was to make model size manageable, not to add additional expressiveness to the CNN over non-CNN models (Gabor models, GLMs); while pooling could possibly introduce some artifacts into our analysis, we thought it was better use it, given that alternative architectures with manageable model size and without pooling—those with a factorized readout layer—performed worse in our pilot experiments.
- Of all the architectures we tried, top-performing ones typically do not have dropout.

We also similarly performed a search over two-convolutional-layer architectures with similar numbers of parameters to that of our baseline 9-channel CNN. The best two-convolutional-layer architecture in our experiments outperformed our one-convolutional-layer baseline marginally (Figure A.5); we chose to focus on one-convolutional-layer CNNs for their simplicity and similarity to non-CNN models (Gabor models, GLMs).

A.1.3 Modeling V1 population using a single CNN

Most CNN-based work models all the neurons in a data set with a single network, with shared parameters in lower layers and separate sets of parameters for different neurons in higher layers [13, 78, 81, 113]. As a preliminary attempt in this direction, we tried modeling monkey A's neurons using a single CNN. The architecture of our CNN is the the same as that in Cadena et al. [13], Klindt et al. [81], with only kernel sizes (kernel size 9 for the first layer and 3 for higher layers) and numbers of channels adapted to our data set. To make the modeling task easier due to the great diversity in our neural data set, we trained two separate CNNs to separately model OT neurons and HO neurons. For each CNN to model a particular neuron subset, we adjusted its numbers of channels so that it has roughly the same number of parameters as all baseline CNN models for that neuron subset collectively; for example, for monkey A's 338 HO neurons, our baseline CNN models have in total $338 \times 883 = 298\,454$ parameters, and our single CNN for HO neurons has correspondingly $296\,308$ parameters with 106 channels for every layer. We ran the training procedure with around 10 sets of regularization hyperparameters for each neuron subset and picked the one with highest testing performance.

To our surprise, given roughly the same number of parameters and within the limit of our hyperparameter tuning, large single CNNs that model all neurons all together performed similarly to our baseline CNNs that model each neuron separately (Figure A.6, top). Our results were in contrast with the intuition that using a single CNN with parameters shared between neurons should increase learning efficiency and model compactness compared to using separate CNNs for different neurons. We also tried turning down model size by changing numbers of channels and found that single CNNs performed worse (Figure A.6, bottom). We also tried other combinations of kernel sizes and numbers of channels, with similar or worse results.

A.1.4 Other output nonlinearities of GLMs

In the main text (Section 3.3.3), we used Poisson GLMs as they are standard in the existing V1 modeling work; in practice Poisson GLMs do not have negative responses which are easier to interpret. We also tried Gaussian and (partially) softplus [46] GLMs (both optimized with mean squared error); we found that the additional exponential nonlinearity in Poisson GLMs was marginally beneficial compared to Gaussian GLMs and softplus GLMs (Figure A.7); using Gaussian or softplus GLMs instead would not change our results in the main text qualitatively.

A.1.5 Other possible stimulus types

Much previous work modeling V1 neurons used natural images or natural movies [13, 25, 78], while we used artificial pattern images [160]. While neural responses to natural stimuli arguably reflect neurons' true nature better, it has the following problems in our current study: 1) public data sets [20] of V1 neurons typically have much fewer images and neurons than our data set, and limited data may introduce bias on the results; 2) artificially generated images can be easily classified and parameterized, and this convenience allows us to classify neurons and compare models over different neuron classes separately (Section 3.2.2.0.1). While white noise stimuli [113, 143] are another option, we empirically found that white noise stimuli (when limited) would not be feasible for finding the correct model parameters (assuming CNN models are correct). We demonstrate this through the following sanity-check experiment: using standard response-triggered (also called spike-triggered for spiking data) methods [122, 144] on a fitted CNN model to recover its ground-truth filters. As shown in Figure A.8, white noise stimuli could not recover ground-truth filters without hundreds of thousands of stimuli, because the neuron, which behaved as a complex pattern detector, was highly nonlinear and highly specific in pattern selectivity, and white noise stimuli would be very inefficient in driving such neurons. The inefficiency of white noise stimuli for identifying convolutional models using was also mentioned in Vintch et al. [172].

A.1.6 More results on pre-trained CNNs

Figure A.9 shows detailed fitting results of the pre-trained VGG19 for the example neurons in Figure 3.7, in a similar format.

Comparing this figure with Figure 3.7, we have two empirical observations as follows.

- pre-trained CNNs (especially their higher layers) had comparable performance as our baseline CNN models for V1 fitting (Figure A.9e vs. Figure 3.7d).
- visualization of these model neurons fitted using pre-trained CNNs is in general difficult. In particular, the complexity of visualized patterns was mostly related to the complexity of the corresponding VGG19 layers where features were extracted, instead of prediction accuracy. For example, while `conv3_1` and `conv4_1` had better performance than `conv2_1`, the visualization of the former two was less intuitive than the latter (c,d vs. b of Figure A.9). We explored different hyperparameters for the visualization and obtained qualitatively similar results.

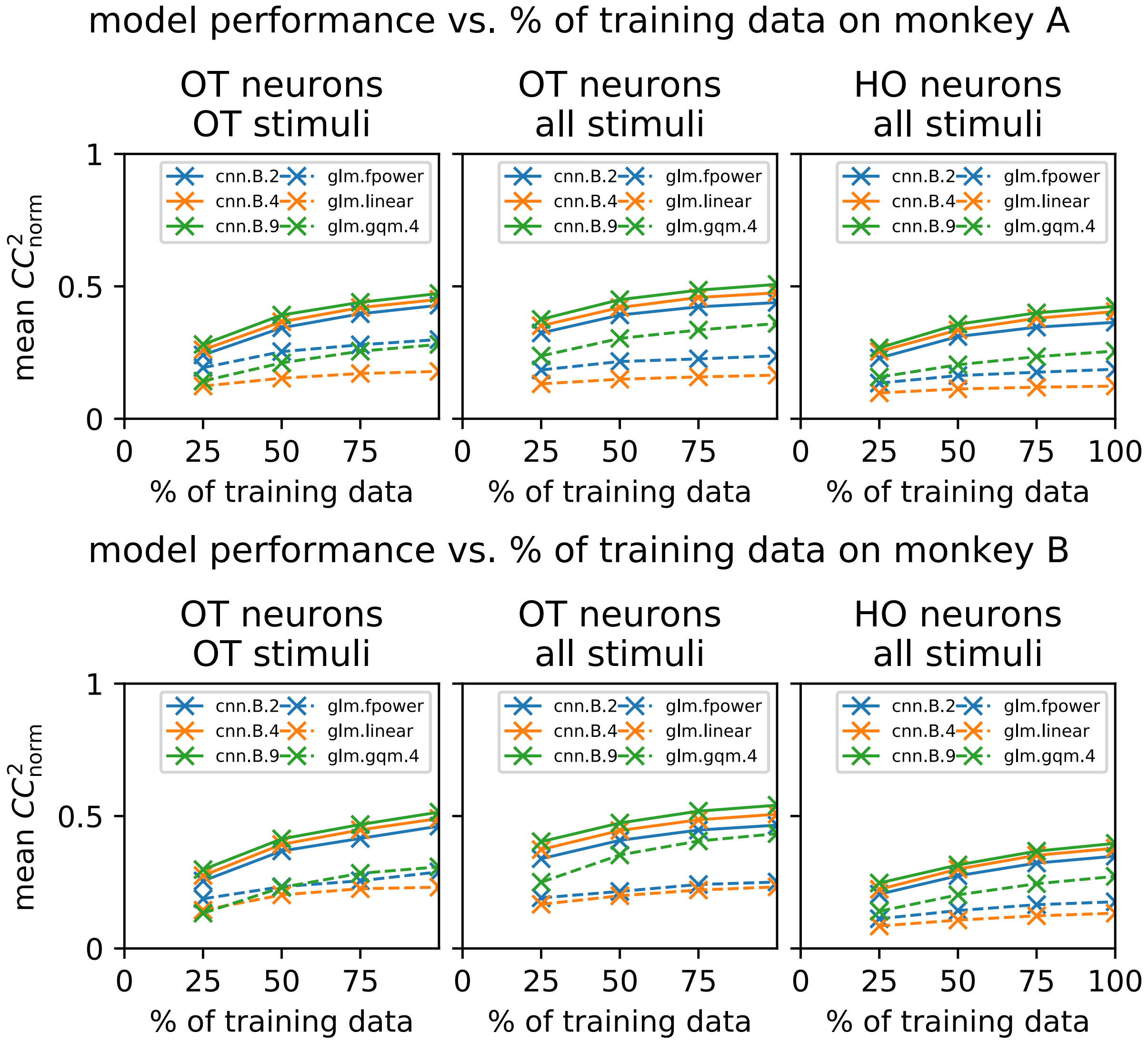


Figure A.1: CNN models vs. others with different amounts of training data. The organization of panels is the same as that in Figure 3.6. For each panel, the model performance is shown in $\mathrm{CC}^2_{\mathrm{norm}}$ averaged over neurons in the neuron subset, as a function of the amount of data used for training and validation. CNN models are shown in solid lines and GLMs are shown in dashed lines; models of similar numbers of parameters share the same color. For GQMs, only the `gqm.4` variant is shown and others gave similar or worse results. Gabor models were not explored due to limited time.

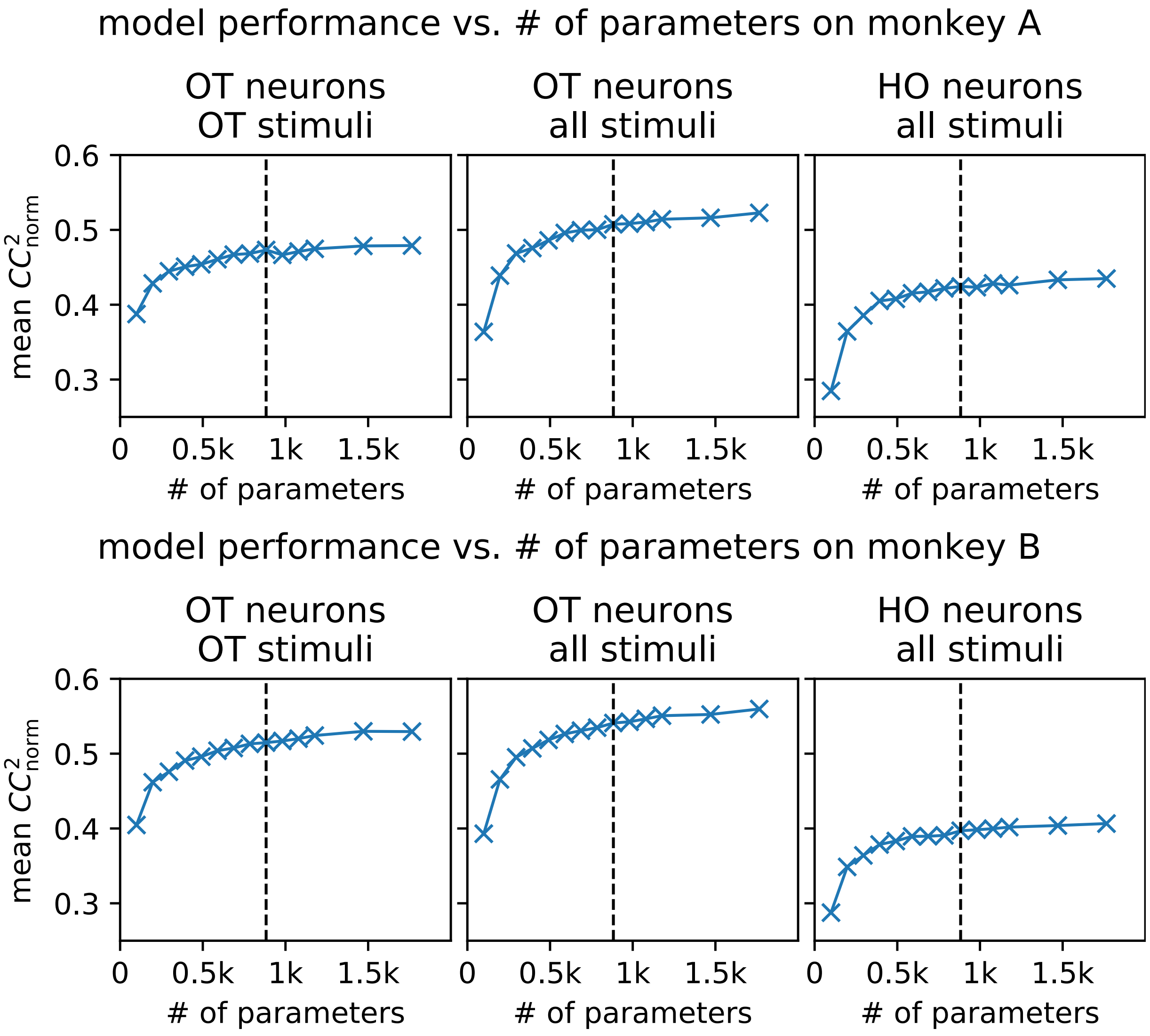


Figure A.2: Performance vs. number of parameters for CNN models. The organization of panels is the same as that in Figure 3.6. For each panel, the model performance is shown in $\mathrm{CC}^2_{\mathrm{norm}}$ averaged over neurons in the neuron subset, as a function of the number of parameters in the model; the vertical line denotes the number of parameters for the baseline CNN model in the main text.

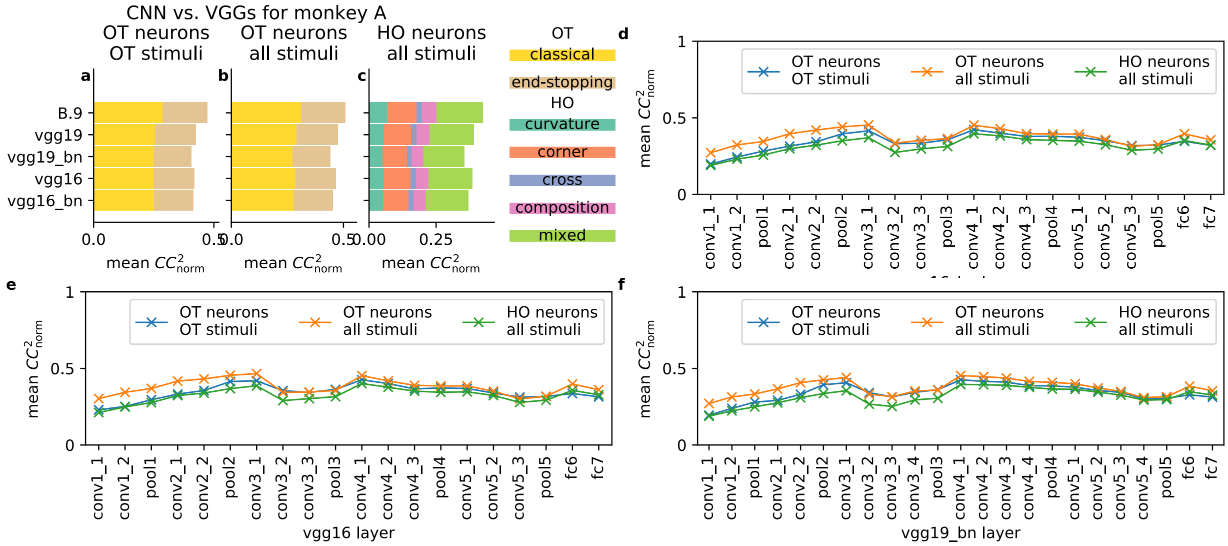


Figure A.3: Transfer learning (goal-driven) approach for modeling V1 neurons using pre-trained VGG networks, monkey A. These panels have similar formats to those in Figure 3.11. **a-c** VGG networks' best performing layers (`conv3_1` for all) vs. the baseline CNN (`B.9`). **d-f** Model performance across layers for different VGG networks (**d** for VGG16 with batch normalization, `e` for VGG16, `f` for VGG19 with batch normalization).

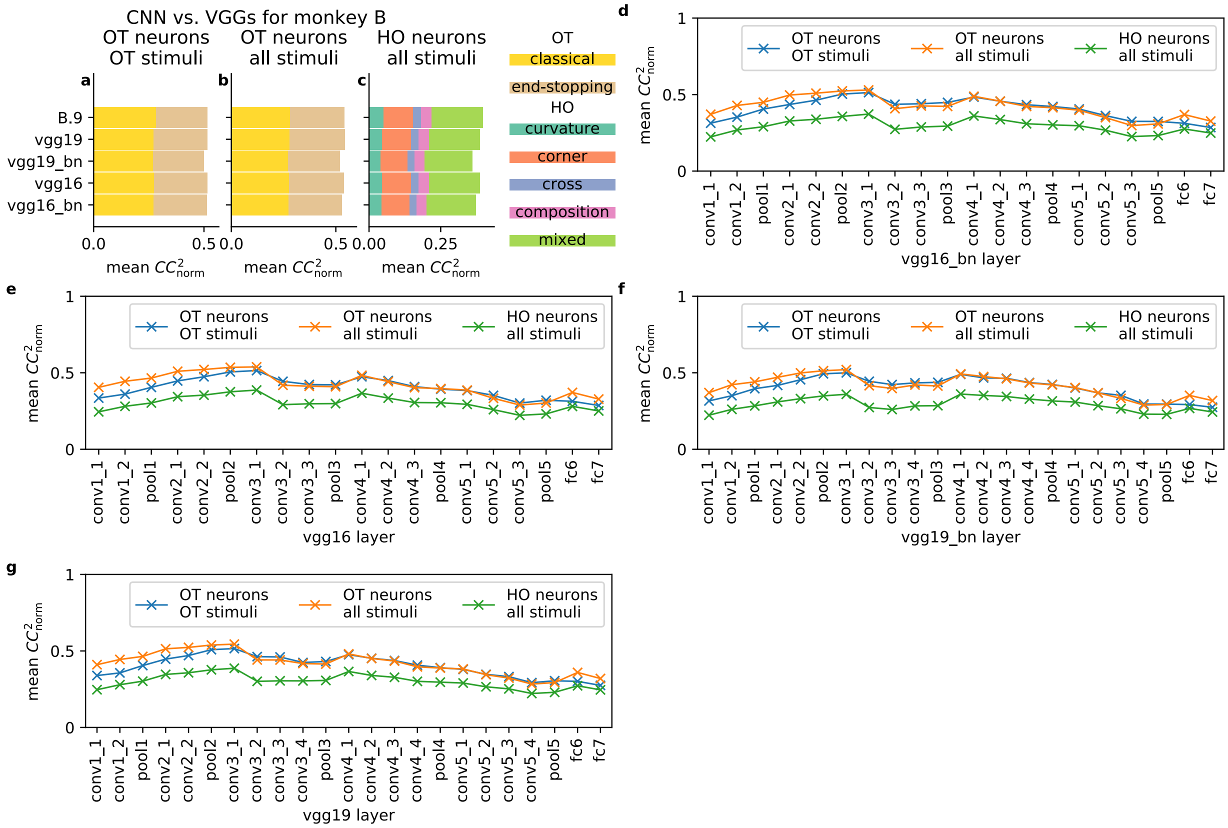


Figure A.4: Transfer learning (goal-driven) approach for modeling V1 neurons using pre-trained VGG networks, monkey B. Compared to Figure A.3, additional VGG19 results (**g**) are shown.

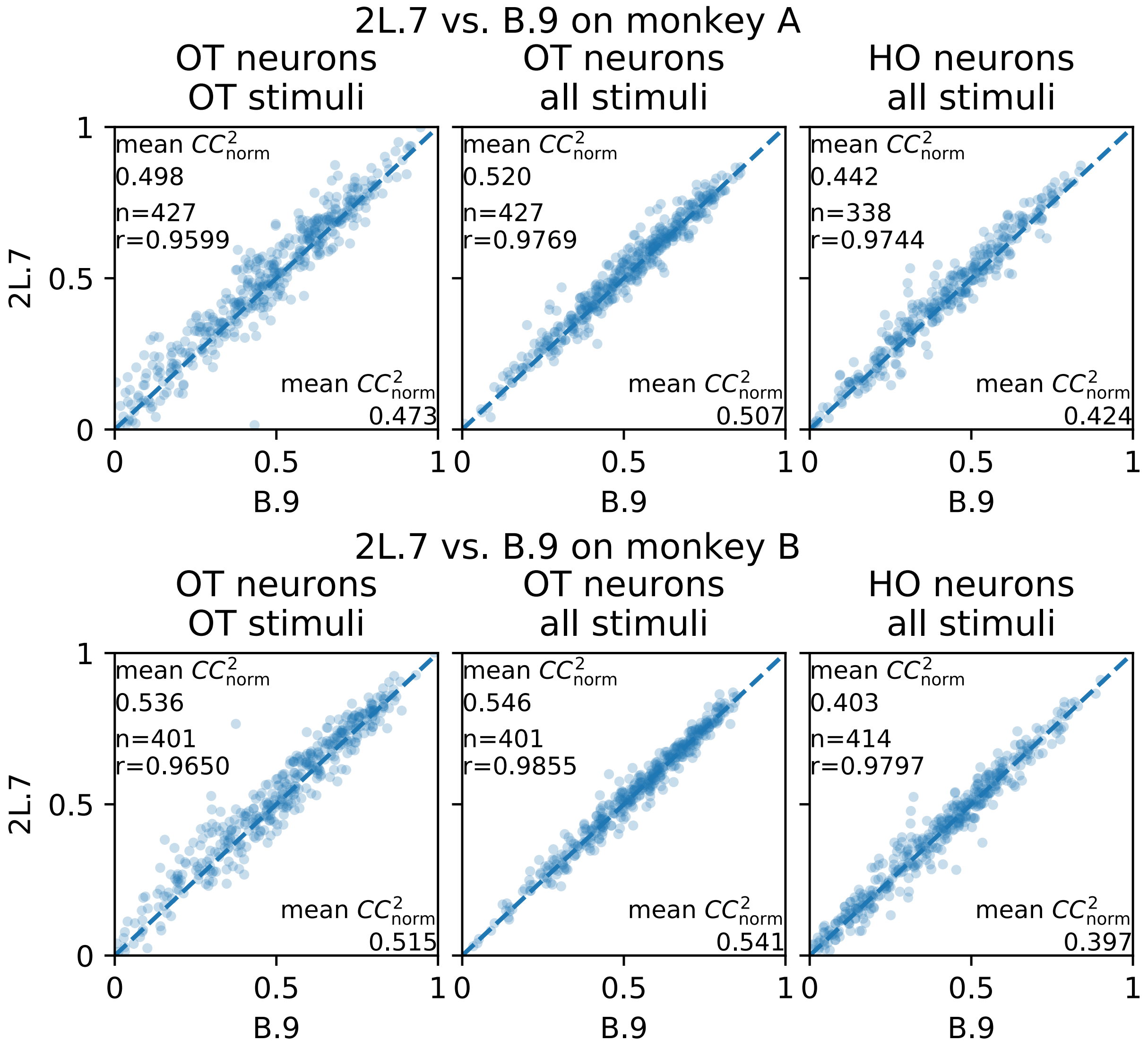


Figure A.5: Neuron-by-neuron comparison of our best candidate two-convolutional-layer CNN (2L.7) vs. the baseline CNN (B.9). The organization of panels is the same as that in Figure 3.6. For each panel, we show the scatter plot of the two models' performance on individual neurons in terms of $\text{CC}^2_{\text{norm}}$ (Section 3.3.6) with performance metrics averaged across neurons at corners and the Pearson correlation coefficient between the two models in the middle. The two-convolutional-layer CNN has 743 parameters with four layers: a convolutional layer with 7 channels and configuration k4d2, a convolutional layer with 7 and configuration k3d1p1, a max pooling layer with configuration k6s2, and finally a fully connected layer; d denotes dilation (set to 1 if omitted) and p denotes padding (set to 0 if omitted).

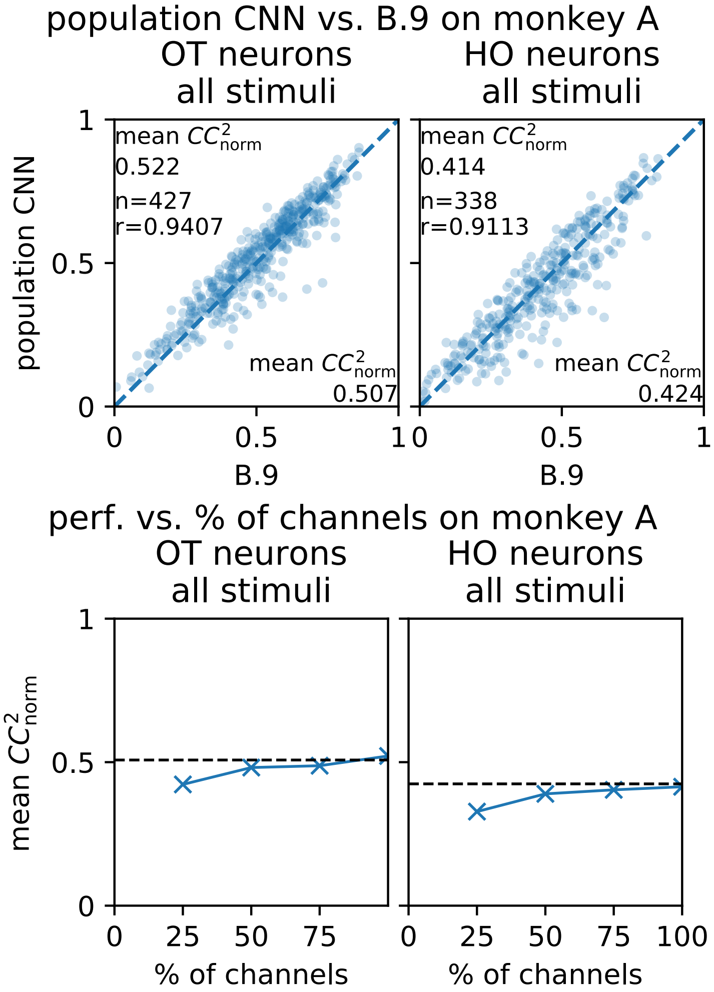


Figure A.6: Performance of modeling multiple neurons using a single CNN, Monkey A. The comparison of our single CNN (`population CNN`) and our baseline CNN (`B.9`) are shown in the top. Only cases for OT neurons, all stimuli and HO neurons, all stimuli were explored due to limited time. For either HO or OT neuron subset, the single CNN has its numbers of channels adjusted to match the set of baseline CNNs for that neuron subset in terms of model size (Section A.1.3). Performance vs. number of parameters for our single CNN are shown in the bottom. Here, number of parameters was changed by reducing the numbers of channels (across all three convolutional layers) of the model to different percentages. Horizontal dashed lines show performance metrics of the baseline CNN.

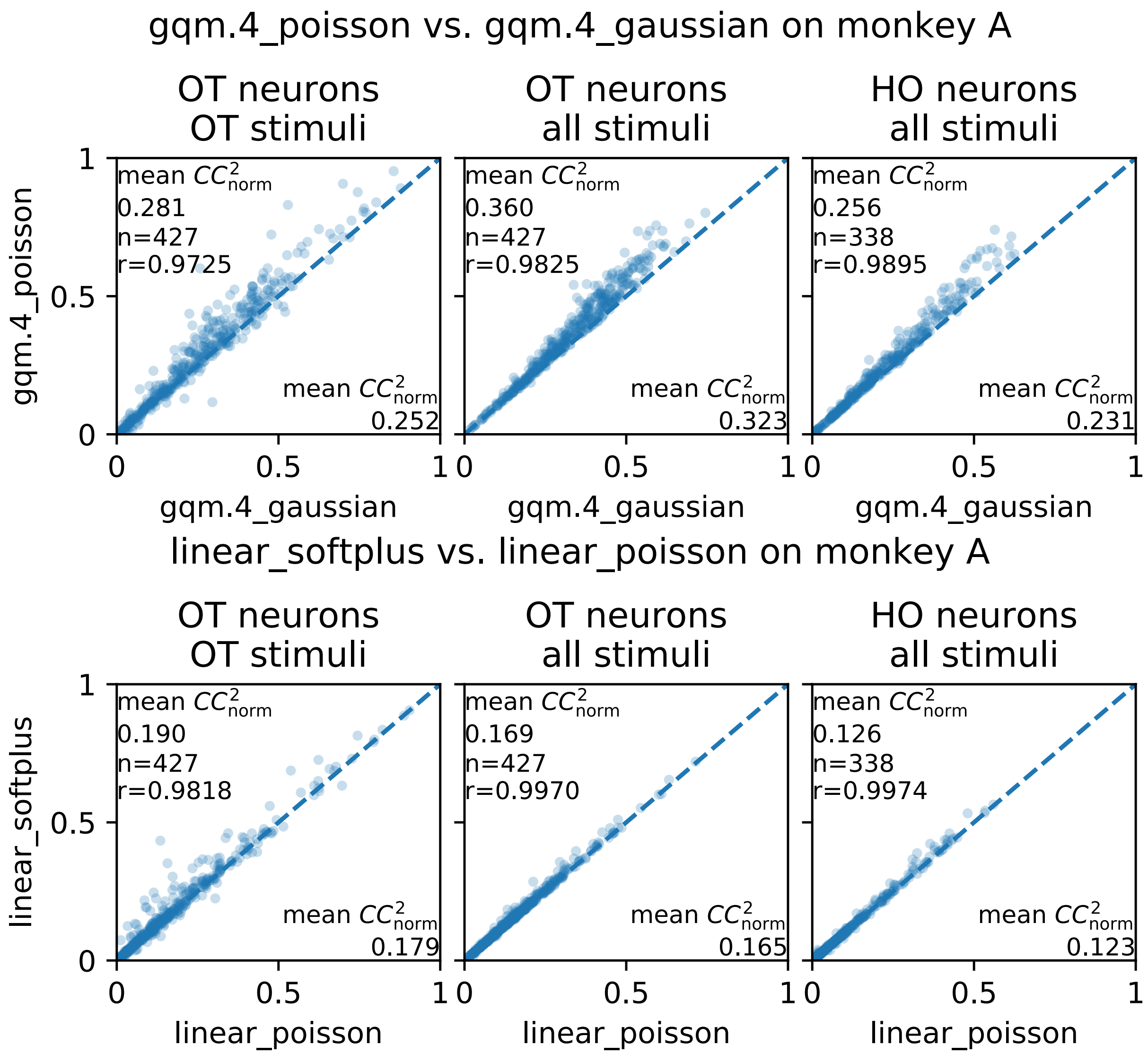


Figure A.7: Neuron-by-neuron comparison of Gaussian, Poisson, and softplus GLMs. The Gaussian and Poisson versions of the GQM with locality 4 (`gqm.4`) were shown in the top, and the Poisson and softplus versions of the vanilla GLM were shown in the bottom. The organization of panels is the same as that in Figure 3.6, except that only results for Monkey A are shown. For each panel, we show the scatter plot of the two models' performance on individual neurons in terms of $\mathrm{CC}^2_{\mathrm{norm}}$ (Section 3.3.6) with performance metrics averaged across neurons at corners and the Pearson correlation coefficient between the two models in the middle. For Gaussian vs. Poisson, results were similar for other GLM variants and Monkey B. For Poisson vs. softplus, we were only able to obtain results for the vanilla GLM on Monkey A due to the slowness of our GLM implementation in MATLAB (`lassoglm`); nevertheless, we believe that the results should generalize to other cases.

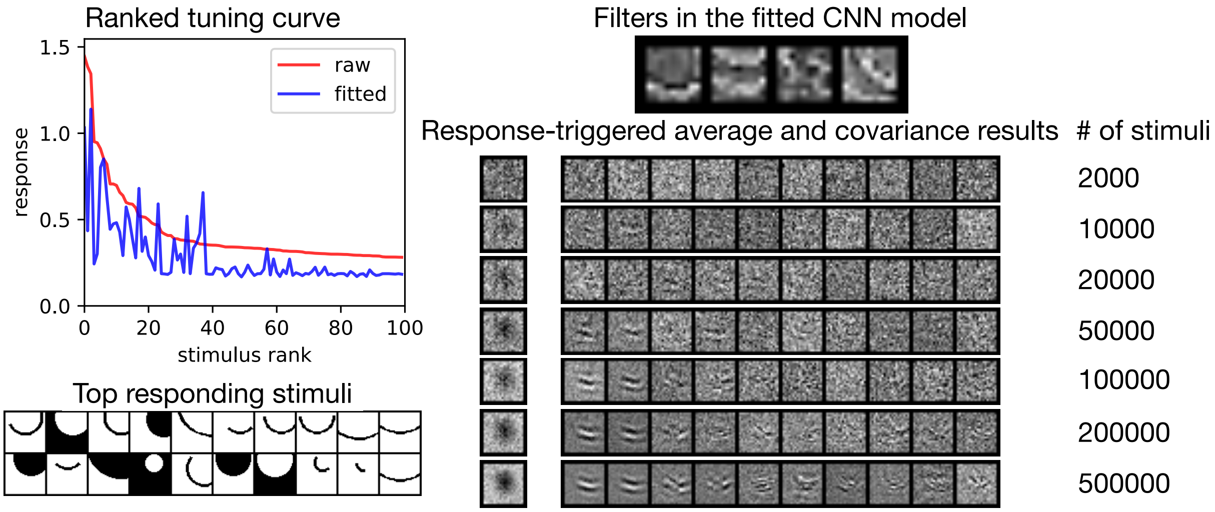


Figure A.8: Failure to recover CNN model parameters using a reasonable amount of white noise stimuli. Left: neuron 554 of monkey A was first fitted by a CNN model with one convolutional layer of four filters (architecture `B.4` in Figure 3.6). Right: standard response-triggered average and covariance methods were applied to the model's responses to different number of Gaussian white noise stimuli to recover its four filters. Apparently, the filters could not be recovered accurately without fewer than 50 000 stimuli, which greatly exceeds the limitation of traditional neurophysiology experiments.

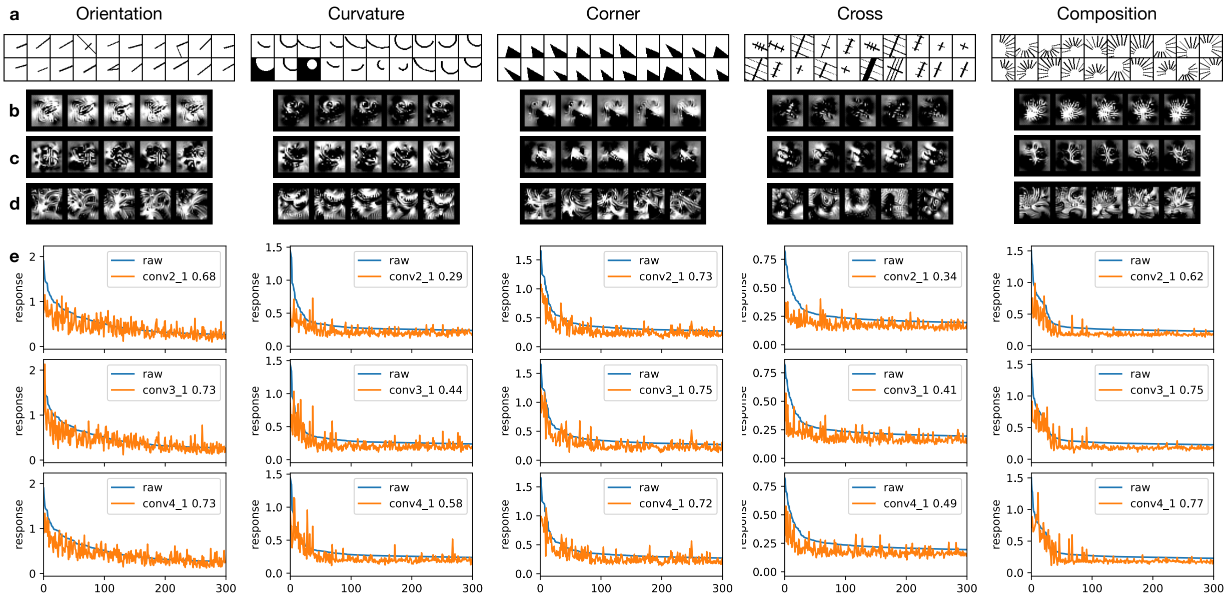


Figure A.9: Example neurons and their fitting results using VGG19. The example neurons are the same as those in Figure 3.7 and two figures have similar formats. For each of the five columns, we show the following information (**a-e**). **a** The top 20 responding stimuli of the neuron; **b-d** visualization results for `conv2_1`, `conv3_1`, and `conv4_1`, respectively; **e** the neuron's fitting results (over testing data) on models fitted using three different VGG19 layers `conv2_1`, `conv3_1`, and `conv4_1`.

Appendix B

Supplementary information for the third study

B.1 Details on neural data sets

B.1.1 Monkey task and recording technique

Data from two series of experiments performed on an awake behaving macaque monkey were used in this study. In both experiments, the monkey performed a fixation task, fixating at a red dot on the screen during a $753\,\mathrm{ms}$ (ImageNet 8K) or $500\,\mathrm{ms}$ (NS 2250) trial. Eye position was monitored continuously with an infrared optical eye tracking system sampling at $120\,\mathrm{Hz}$ (ISCAN). A trial was aborted without reward if, at any point prior to delivery of reward the monkey failed to maintain fixation within a central window spanning $0.6°$ to $0.8°$. We monitored neuronal activity through an SC96 array, a modular, replaceable micromanipulator system allowing independent bidirectional control of 32 microelectrodes arranged in a square array with $1.5\,\mathrm{mm}$ inter-electrode spacing (Gray Matter Research, MT). The array was implanted over the intact dura above the occipital operculum with its center over areas V1 and V2. A screw-driven mechanism allowed independent bi-directional control of the depth of each electrode over a range of $16\,\mathrm{mm}$ with an accuracy of approximately $15\,\mathrm{\mu m}$. This provided sufficient control to isolate the spiking activity of individual neurons. The location of the tip of each electrode remained relatively stable across multiple days as evidenced by consistency in the pattern of neuronal selectivity for familiar images. We used the same sampling and filtering hardware, software, and settings that we used for the Utah Intracortical Array in our earlier experiments with the Blackrock Cerebus multi-channel Neural Data Acquisition System. We used the same spike sorting procedures to isolate single or multi-unit waveforms [74]. The only selection criterion that we applied was that the unit had to have to exhibit significant average PSTH relative to baseline averaged across all stimuli. In total, we collected 79 neurons for ImageNet 8k experiment and 34 for NS 2250.

B.1.2 Stimulus preparation and presentation

B.1.2.1 Stimulus presentation details

We displayed stimuli on a calibrated CRT monitor ($85\,\mathrm{Hz}$ frame rate, around $40\,\mathrm{cd\,m^{-2}}$ mean luminance) placed $100\,\mathrm{cm}$ from the animal, using NIH CORTEX software. Two different stimulus presentation paradigms were used. In the ImageNet 8K experiment, 8000 stimuli were presented in 16 stimuli per trial while the monkey fixated. Each trial lasted for $753\,\mathrm{ms}$, and thus each image was presented for $47\,\mathrm{ms}$ (4 frames at $85\,\mathrm{Hz}$). The 8000 stimuli were presented once each in a 500-trial run. Each day, 3000 trials were typically acquired from the monkey, resulting in 6 repetitions per stimulus. Each image would appear in different image sequences and at different position in the sequence in different runs. Three recording sessions were performed across six days. In the NS2250 experiment, a total of 2250 stimuli were tested. Each image was presented for $500\,\mathrm{ms}$, and repeated 8 to 10 times randomly interleaved with other stimuli. Each day, 500 images were tested in 4000 trials. Each day 25 calibration images were also tested to track neurons across days based on their stimulus response finger-print. The recording of response to 2250 stimuli were obtained in five consecutive days.

B.1.2.2 Stimulus preparation details

B.1.2.2.1 ImageNet 8K We used natural images from ImageNet [141]. We converted color images to gray scale ones, and extracted $25\,000$ patches of $256\,\mathrm{px}$ by $256\,\mathrm{px}$. Each degree visual angle corresponds to around $25\,\mathrm{px}$ on the monitor. Out of the $25\,000$, 8000 image patches were selected for which the average pixel values were in the range 40–200 out of 0-255 and we excluded images of low RMS contrast (≤ 0.25) at the central 3° visual angle region of the images which covered the receptive fields of most of the neurons monitored by the array. The stimuli were presented in a $10°$ diameter circular aperture within a gray surround. The central $200\,\mathrm{px}$ by $200\,\mathrm{px}$ part used for modeling (Section B.3.1.1) has an average RMS contrast of 0.3.

B.1.2.2.2 NS 2250 We used the 2250 stimuli extracted from images used in Tang et al. [161]. We converted color images to gray scale values. Each image was presented within a 8° aperture ($200\,\mathrm{px}$). The central $148\,\mathrm{px}$ by $200\,148$ part used for modeling (Section B.3.2.1) has an average RMS contrast of 0.27.

B.1.3 Extraction of responses to the ImageNet 8K stimuli

While the extraction of the responses to stimuli in the NS2250 stimuli was straightforward and standard, the extraction of the responses to the ImageNet 8K stimuli was more challenging for two difficulties: first, each stimulus was presented for $47\,\mathrm{ms}$ and thus the responses to the 16 consecutive distinct stimuli presented in a trial were superimposed onto each other; second, there was a delay between stimulus onset and the response onset of V1 and V2 neurons, and the delay was slightly different for each neuron ($30\,\mathrm{ms}$ to $50\,\mathrm{ms}$). The first problem turned out to be not severe because neural responses to natural stimuli tended to be sparse, typically yielding 0 to 3 spikes for each stimulus' $47\,\mathrm{ms}$ presentation and neurons typically did not respond well to consecutive stimuli, mitigating the potential response superposition issue. To resolve the second problem, for each neuron, we selected a different offset for response calculation so that the pairwise correlations of neural responses across different recording sessions were maximized and thus there was a different offset for each neuron.

B.2 Detailed description of models explored in this study

B.2.1 Baseline models

During inference, a baseline model takes an image $\boldsymbol{x} \in \mathbb{R}^{H \times W}$ of height H and width W as input and generates an prediction $\vec{\hat{\boldsymbol{r}}} \in \mathbb{R}^N$ of the neural responses $\vec{\boldsymbol{r}} \in \mathbb{R}^N$ to the image. Mathematically, the model inference process is defined by Eqs. (B.1).

$$\begin{aligned}
\vec{\hat{\boldsymbol{r}}} &= \mathrm{Act}(\mathrm{FC}(\mathrm{Pool}(\boldsymbol{y}^{(M)}))) && \text{(B.1a)}\\
\boldsymbol{y}^{(m)} &= \mathrm{CPB}^{(m)}(\boldsymbol{y}^{(m-1)}) \quad m = 2, \ldots, M && \text{(B.1b)}\\
\boldsymbol{y}^{(1)} &= \mathrm{CPB}^{(1)}(\mathrm{BatchNorm}(\boldsymbol{x})) && \text{(B.1c)}\\
\mathrm{CPB}^{(m)}(\boldsymbol{y}) &= \mathrm{Act}(\mathrm{BatchNorm}^{(m)}(\mathrm{Conv}^{(m)}(\boldsymbol{y}))) && \text{(B.1d)}
\end{aligned}$$

For a model of M convolutional processing blocks (CPBs), the model inference starts with Eq. (B.1c) to normalize the input image and obtain the initial CPB's output $\boldsymbol{y}^{(1)}$, followed by a few applications of Eq. (B.1b) to get outputs of later CPBs $\boldsymbol{y}^{(2)}, \ldots \boldsymbol{y}^{(M)}$. Finally, Eq. (B.1c) is applied to obtain the predicted neural responses $\vec{\hat{\boldsymbol{r}}}$. Conv, $\mathrm{BatchNorm}$, Act, Pool, FC represent convolution, batch normalization, nonlinear activation, average pooling, and factorized fully connected layers [81], respectively. Different convolutional processing blocks have different hyper parameters and learned parameters, as denoted by superscripts (m) in Eq. (B.1d).

B.2.2 Recurrent models

The model inference process is defined by Eqs. (B.2).

$$\vec{\hat{\boldsymbol{r}}}_{\texttt{no-avg}} = \mathrm{Act}(\mathrm{FC}(\mathrm{Pool}(\boldsymbol{y}^{(M,T)}))) \tag{B.2a}$$

$$\vec{\hat{\boldsymbol{r}}}_{\texttt{early-avg}} = \mathrm{Act}(\mathrm{FC}(\mathrm{Pool}(\overline{\boldsymbol{y}}^{(M,T)}))) \tag{B.2b}$$

$$\vec{\hat{\boldsymbol{r}}}_{\texttt{late-avg}} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{Act}(\mathrm{FC}(\mathrm{Pool}(\boldsymbol{y}^{(M,t)}))) \tag{B.2c}$$

$$\vec{\hat{\boldsymbol{r}}}_{\texttt{2-avg}} = \frac{1}{T}\sum_{t=1}^{T} \mathrm{Act}(\mathrm{FC}(\mathrm{Pool}(\overline{\boldsymbol{y}}^{(M,t)}))) \tag{B.2d}$$

$$\overline{\boldsymbol{y}}^{(M,t)} = \frac{1}{t}\sum_{t'=1}^{t} \boldsymbol{y}^{(M,t')} \tag{B.2e}$$

$$\boldsymbol{y}^{(m,t)} = \mathrm{RCPB}^{(m,t)}(\boldsymbol{y}^{(m-1,t)}, \boldsymbol{y}^{(m,t-1)}) \quad m = 2, \dots M; t = 1, \dots T \tag{B.2f}$$

$$\boldsymbol{y}^{(1,t)} = \mathrm{RCPB}^{(1,t)}(\mathrm{BatchNorm}(\boldsymbol{x}), \boldsymbol{y}^{(1,t-1)}) \quad t = 1, \dots T \tag{B.2g}$$

$$\boldsymbol{y}^{(m,0)} = \boldsymbol{0} \tag{B.2h}$$

$$\mathrm{RCPB}^{(m,t)}(\boldsymbol{y}, \boldsymbol{y}') = \mathrm{Act}(\mathrm{BatchNorm}^{(m,t)}(\mathrm{Conv}^{(m)}_{\text{feed-forward}}(\boldsymbol{y}) + \mathrm{Conv}^{(m)}_{\text{lateral}}(\boldsymbol{y}'))) \tag{B.2i}$$

For a model of M recurrent convolutional blocks (CPBs) and T iterations in total, the model inference starts with T cycles of Eqs. (B.2g),(B.2f) to obtain the responses of all M RCPBs across T iterations $\boldsymbol{y}^{(m,t)}, m = 1 \dots M, t = 1 \dots T$. Finally, one of Eqs. (B.2a),(B.2b),(B.2c),(B.2d) is used to obtain the final model output depending on the readout mode used (Section 4.3.1.2.2).

There are two details worth noting about our implementation of the recurrent model. First, RCPBs are technically not fully recurrent. As indicated by the notations in Eq. (B.2i), for a given RCPB, the convolution layers are fixed across iterations but different batch normalization layers are used for different iterations t. We use different batch normalization layers to accommodate different mean and variance statistics of responses across different iterations, with minimal increase in model size ($\leq 1\%$). Second, a RCPB might have fewer iterations T' than the overall model iteration T. For example, a CPB is a RCPB with $T' = 1$ iteration; there is no lateral convolutional layer or additional batch normalization layer to define model output $\boldsymbol{y}^{(t)}, t > 1$. In such cases, we use the output at iteration T' as the output for later iterations, or $\boldsymbol{y}^{(t)} = \boldsymbol{y}^{(T')}, t = T' + 1, \dots, T$; with such definition, Eq. (B.2) are applicable to recurrent models explored in this study, where each recurrent model has one CPB followed by a number of RCPBs.

B.3 Details an data preprocessing

B.3.1 ImageNet 8K

B.3.1.1 Input

Each original image was of size $400\,\mathrm{px}$ x $400\,\mathrm{px}$ with an aperture of around $256\,\mathrm{px}$ in diameter. We cropped out the central $200\,\mathrm{px}$ x $200\,\mathrm{px}$ portions and downsampled them to $50\,\mathrm{px}$ x $50\,\mathrm{px}$ (4x) as the input. The reason we used $200\,\mathrm{px}$ x $200\,\mathrm{px}$ portions was that these portions contained actual image pixels only and without apertures, which are standard in neurophysiological experiments and may have undesirable effects in data-fitting due to the sharp difference between pixels inside and outside the aperture. We downsampled the images to reduce model size and follow similar practices in recent related studies [13, 78].

B.3.1.2 Output

Given a neuron, for each image in a recorded trial, we computed the spike count during the time window of around $47\,\mathrm{ms}$ when the image was presented on the screen, with some offset (B.1.3) to account for the delay between stimulus onset and the neural response. We then averaged the spike counts over all trials to obtain the average spike count for this neuron for this image. We repeated the above procedure for every neuron and image to obtain 8000 average spike counts for each of the 79 neurons. Finally, for each of the 79 neurons, we scaled its responses so that the average over all 8000 images was 0.5, following practices in the literature [78]. The scaled average spike counts were used as the ground truth output.

B.3.2 NS 2250

B.3.2.1 Input

Each original image was of size $252\,\mathrm{px}$ x $252\,\mathrm{px}$ with an aperture of around $200\,\mathrm{px}$ in diameter. As in the data preprocessing of ImageNet 8K data, we cropped out the central $148\,\mathrm{px}$ x $148\,\mathrm{px}$ portions and downsampled them to $37\,\mathrm{px}$ x $37\,\mathrm{px}$ (4x).

B.3.2.2 Output

As in the data preprocessing of ImageNet 8K data, we obtained 2250 average spike counts for each of the 34 neurons, by collecting spike counts during the $500\,\mathrm{ms}$ window of stimulus presentation with some offset. For each of the 34 neurons, we scaled its responses so that the average over all 2250 images was 0.5. The scaled average spike counts were used as the ground truth output.

While results shown in this study for this data set used spike counts during the full $500\,\mathrm{ms}$ window, we also tried using shorter windows (first $100\,\mathrm{ms}$, last $100\,\mathrm{ms}$, etc.); recurrent models outperformed baseline feed-forward ones regardless of the choice. We chose to present results based on the full $500\,\mathrm{ms}$ window because the the average spike counts from the full window could be used to decode the input images with an accuracy higher than other shorter windows.

Following Tang et al. [161], we used the `NearestCentroid` classifier implemented in scikit-learn [127] to compute the decoding accuracy. For a given time window (say, the full $500\,\mathrm{ms}$ one), we computed the cross-validated classification accuracy based on the neural responses to all 2250 images with 8 trials per image (for images with 9 or 10 trials, we only kept the first eight trials in this experiment for simplicity, as all images had at least eight trials). We performed the cross validation over trials for eight times, each time using data over seven trials for training and those over one trial for testing; during training, the decoder simply computed the average response over seven trials for each image; during testing, the decoder classified the input neural response vector to the image whose average neural response stored in the decoder was closest to the input neural response vector. Table B.1 gives the performance of the decoder when different time windows were chosen.

Table B.1: Image decoding accuracies of neural responses using different time windows.

time window	cross-validated decoding accuracy
$0\,\mathrm{ms}$ to $500\,\mathrm{ms}$	71.8%
$0\,\mathrm{ms}$ to $400\,\mathrm{ms}$	70.4%
$100\,\mathrm{ms}$ to $500\,\mathrm{ms}$	61.3%
$0\,\mathrm{ms}$ to $100\,\mathrm{ms}$	55.2%
$100\,\mathrm{ms}$ to $200\,\mathrm{ms}$	40.6%
$200\,\mathrm{ms}$ to $300\,\mathrm{ms}$	28.1%
$300\,\mathrm{ms}$ to $400\,\mathrm{ms}$	19.4%
$400\,\mathrm{ms}$ to $500\,\mathrm{ms}$	14.6%

B.4 Model performance evaluation in detail

Given a trained model, we use average $\mathrm{CC}^2_{\text{norm}}$ over all neurons to quantify its performance on a data set. For each neuron, we compute its $\mathrm{CC}^2_{\text{norm}}$ based on Eqs. (B.3).

$$\mathrm{CC}^2_{\text{norm}} = \frac{\mathrm{CC}^2_{\text{raw}}}{\mathrm{CC}^2_{\text{max}}} \tag{B.3a}$$

$$\mathrm{CC}_{\text{raw}} = \mathrm{Pearson}(\vec{\boldsymbol{r}}, \hat{\vec{\boldsymbol{r}}}) \tag{B.3b}$$

$$\mathrm{CC}_{\text{max}} = \sqrt{\frac{\mathrm{Var}(\{\sum_k r_{m,k}\}) - \sum_k \mathrm{Var}(\{r_{m,k}\})}{K(K-1)\mathrm{Var}(\{r_m\})}}. \tag{B.3c}$$

$$\vec{\boldsymbol{r}} = (r_1, r_2, \ldots, r_M) \tag{B.3d}$$

$$r_m = \frac{\sum_{k=1}^{K} r_{m,k}}{K} \tag{B.3e}$$

Concretely, we first compute the raw Pearson correlation CC_{raw} between the ground truth trial-averaged neural responses $\vec{\boldsymbol{r}}$ as defined by Eqs. (B.3d),(B.3e) and the model responses $\hat{\vec{\boldsymbol{r}}}$

using Eq. (B.3b). We then divide CC_{raw} by CC_{max}, which is defined in Eq. (B.3c) and estimates the maximal Pearson correlation coefficient an ideal model can achieve given the noise in the neural data [60, 149]; inside the square root, the numerator is the difference between variance of response sums per stimulus $\sum_k r_{m,k}$ and sum of response variances per trial $\mathrm{Var}(\{r_{m,k}\})$, and the denominator is the variance of the trial-average neural responses scaled by a factor related to the number of trials K. Finally, we get the square of normalized Pearson correlation coefficient using Eq. (B.3a). As squared CC_{raw} gives the fraction of variance in neural responses explained by the model in a simple linear regression, squared $\mathrm{CC}_{\text{norm}}$ gives the normalized explained variance that accounts for noise in the neural data. Note that we used neural responses on all stimuli instead of on testing set stimuli to compute CC_{max} in Eq. (B.3c) for more accurate estimation. In addition, because the NS2250 data set has a variable number of trials per image (8 to 10), we only used the first eight trials for simplicity.

B.5 Multi-path models as approximations of recurrent models

In Section 4.4.2, to demonstrate the validity of multi-path reformulation (Section 4.3.4), we converted each recurrent model into a corresponding multi-path model following the procedure illustrated in Figure 4.5 and then retrained the multi-path model from scratch under the same hyperparameters as the original recurrent model. Unlike the recurrent model which combines multiple information flow paths using shared components (Figure 4.5a,b), the multi-path model has separate feed-forward paths (Figure 4.5c,d) with shared convolutional weights. Results are shown in Figure 4.13. Note that the total number of multi-path models (sum of all n's in the figure) was lower than the total number of recurrent models we had, because multi-path models took more GPU memory and resulted in out of memory for larger models like those with three layers and more than 32 channels.

In addition, note that the retrained multi-path models has shared parameters across paths for convolutional layers but separate parameters for batch normalization layers. While sharing batch normalization layers results in a more faithful reformulation of the original recurrent model, in practice we found that separate BN parameters gave much better performance at a cost of small model size increase and sharing BN parameters made multi-path models perform much worse as each BN layer has to learn the statistics over multiple paths in this setting.

In addition to neural data prediction performance (Figure 4.13), we also found that recurrent models and the corresponding multi-path ones were similar in terms of other statistics, such as average path length (Figures 4.14,4.15; Section 4.4.3).

B.6 Computation of average path length and path diversity of a recurrent model

Given a trained recurrent model, we define its average path length and path diversity based on its recurrent block(s) and readout mode. For a model with T iterations, there are T ensembles

(Figure 4.5c,Figure 4.5d), each corresponding to the information flow of recurrent block(s) at a particular iteration (Figure 4.5a,Figure 4.5b). We then perform the following steps to get the average path length and path diversity for the recurrent model of a particular readout mode.

1. Determine the *strength score* of every component in every path for every ensemble (Figure 4.16a). The definition of the score is designed so that the *strength score* of every component is the ratio of the magnitude of the component's output over that of its input, when the input is a single scalar (thus, the convolutional layer's kernel size is 1 by 1, and there is only one channel per layer). This corresponds to Figure 4.16a.
   (a) The *strength score* of a BN layer is the average of the absolute values of the scaling factors used during inference, over all output channels.
   (b) The *strength score* of a convolutional layer is the average of the 2-norms of 3-D convolutional kernels flatten into vectors, over all output channels.
   (c) The *strength score* of an activation layer (ReLU or softplus in this study) is 1, as the activation layer outputs the input itself when the input is a large enough positive number.
2. Compute the *unnormalized strength* of each path (Figure 4.16b). The unnormalized strength of each path is the product of strength scores of all its components, further multiplied by a weight factor determined by the readout mode. The extra weight factor is needed because different readout types by definition utilize paths in different iterations differently (Section 4.3.1.2.2). For example, the no-avg readout mode only uses paths in the last iteration, whereas early-avg uses all paths over all iterations equally. The (relative) value the weight factor can be easily calculated based on the definition of different readout modes (Section 4.3.1.2.2).
3. Compute the *normalized strength* of each path and compute average path length and path diversity (Figure 4.16c). The normalized strength of each path is proportional to its unnormalized strength so that all normalized strengths over all paths sum to 1.
   (a) The *average path length* is the weighted average of all paths' lengths, weighted by the paths normalized strengths. The length of a path is defined as the number of convolutional layers in the path. Equivalently, it can be also defined as the number of BN or activation layers in the path, as a path has the same number of convolutional layers as that of BN layers and as that of activation layers.
   (b) The *path diversity* is the path length distribution, where the probability of each path length is the sum of normalized strengths of paths of that length. Note that multiple paths may have the same length, as demonstrated in Figures 4.5b,4.5d.

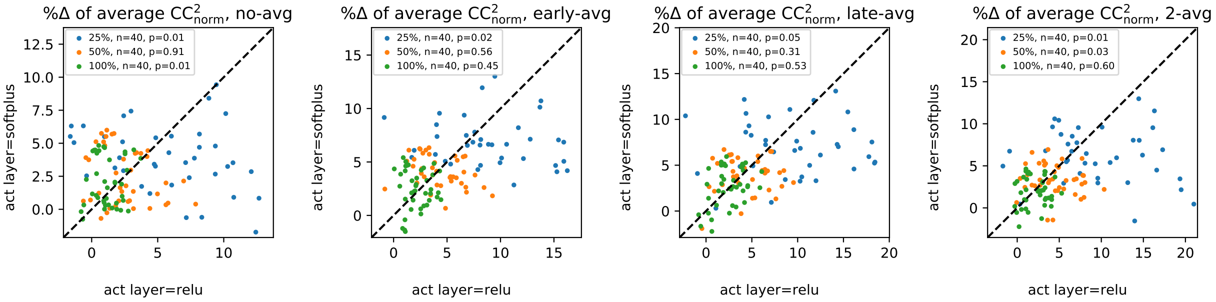
B.7 Ablation study

To compute the baseline shown as horizontal dashed lines in Figures 4.21,4.22, we trained the multi-path models corresponding to the original recurrent models with 2 layers, 7 iterations, and 16 or 32 channels using 100% of training data. The construction of multi-path models and their subtle differences from the original recurrent models are described in Appendix B.5. We used all the training data because the reformulation of recurrent models as multi-path models worked better with more training data (Figures 4.13,4.14,4.15); we did not train recurrent models with other numbers of layers or channels because models with 3 layers or more than 32 channels exceeded GPU memory limits during training and 8-channel recurrent models did not outperform feed-forward models much (Figures B.6,B.9). In addition, for the ImageNet 8K data, eight multi-path models exceeded GPU memory limits during training, causing n, the number of models per data point to be 24 in Figures 4.21 instead of 32 in Figures 4.22.

To compute the performance metrics of ablated models in the figures, we trained additional multi-path models like those for getting the baseline, but with paths of certain lengths removed. In each of the figures, the left panel shows the performance metric change when paths of a particular length (1,2,3,4,5,6) were removed; the middle panel shows the performance metrics change when paths of two adjacent lengths (1-2, 2-3, 3-4, 4-5, 5-6) were removed; the right panel shows the performance metric change when paths of three adjacent lengths (1-3, 2-4, 3-5, 4-6, 5-7) were removed.

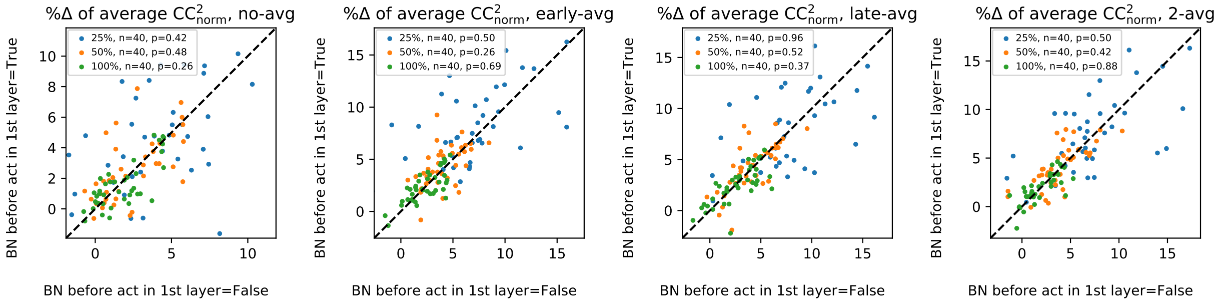
B.8 Additional results on recurrent vs. feed-forward models

B.8.1 Results under different model size-independent hyperparameters



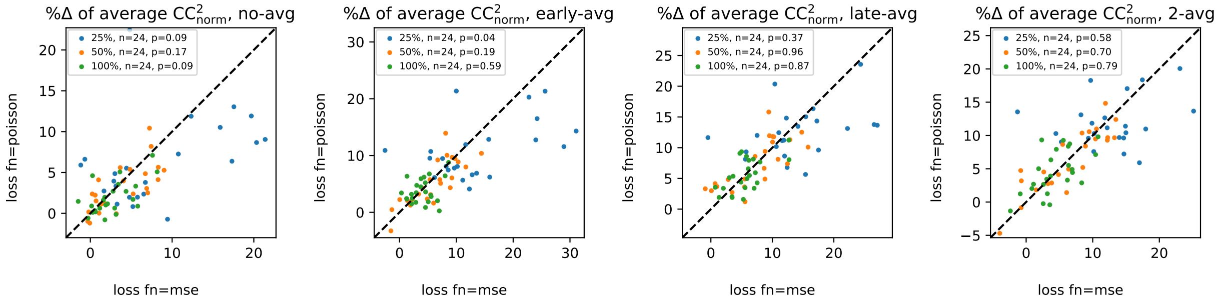
(a) (b) (c) (d)

Figure B.1: The relative performance metric changes of recurrent models over feed-forward models under different activation layers, on the ImageNet 8K data set. See Figure 4.8 for details.



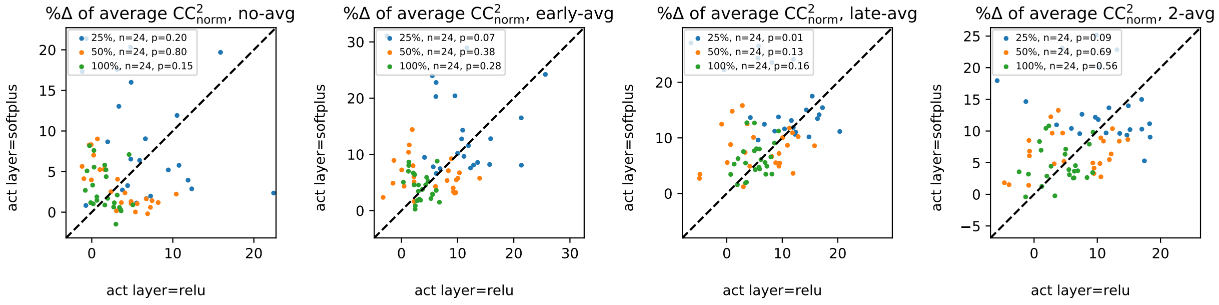
(a) (b) (c) (d)

Figure B.2: The relative performance metric changes of recurrent models over feed-forward models under different orders of BN and activation in the first convolutional block, on the ImageNet 8K data set. See Figure 4.8 for details.



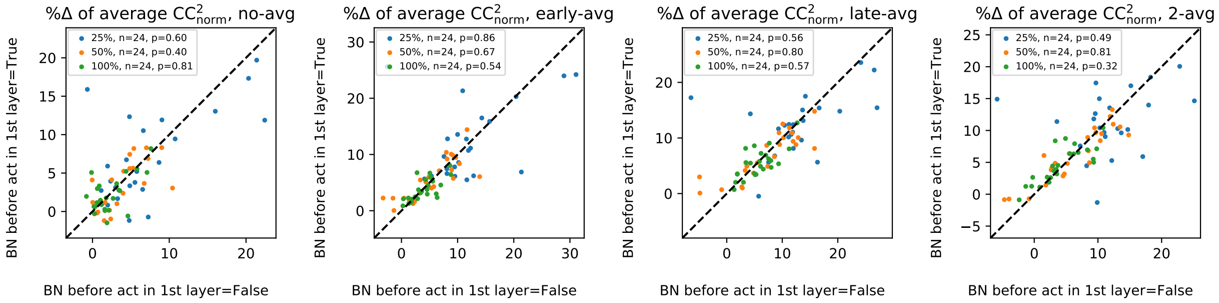
(a) (b) (c) (d)

Figure B.3: NS 2250 version of Figure 4.8.



(a) (b) (c) (d)

Figure B.4: NS 2250 version of Figure B.1.



(a) (b) (c) (d)

Figure B.5: NS 2250 version of Figure B.2.

B.8.2 Results grouped by model size and amount of training data

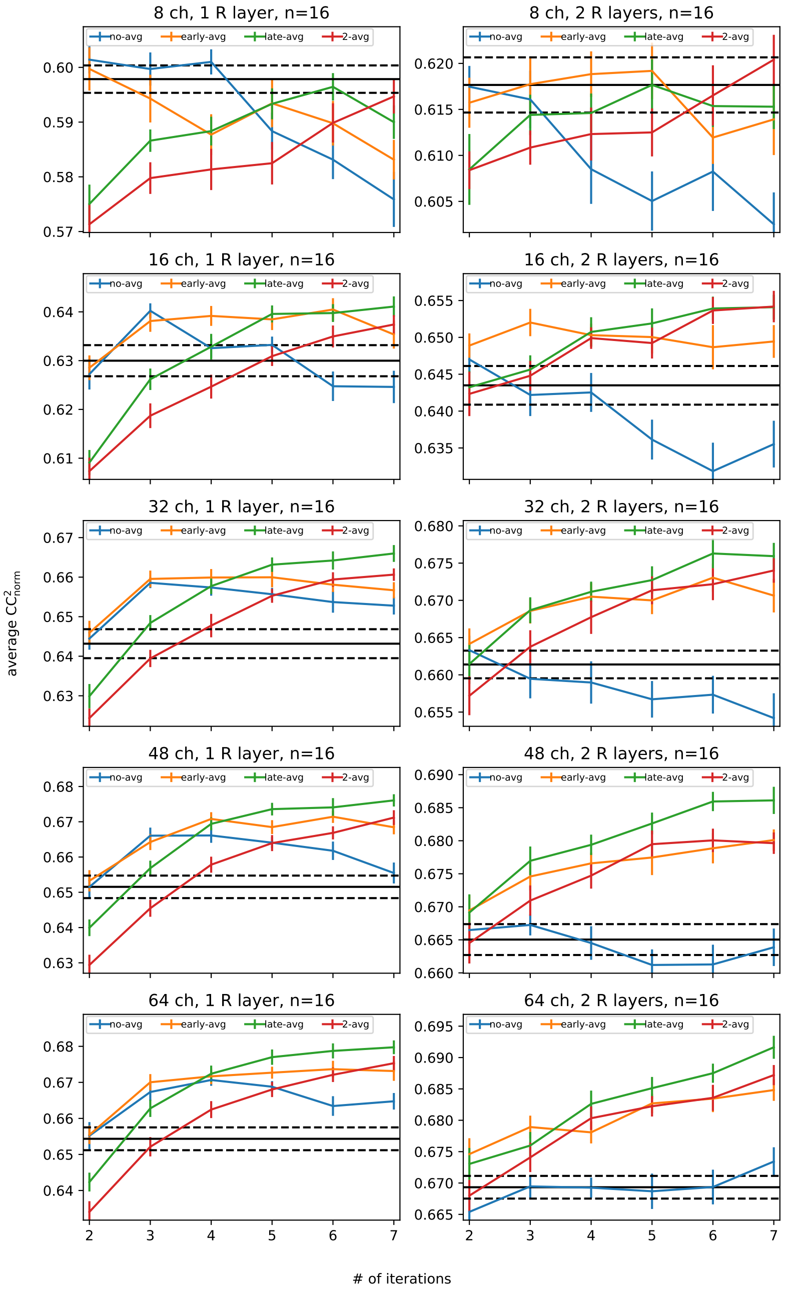


Figure B.6: Recurrent models vs. feed-forward models on the ImageNet 8K data, using 100% of available training data, under different model sizes by changing number of channels and number of layers. It's a more detailed version of Figure 4.11. For each panel, its title denotes the number channels (ch) and the number of recurrent layers.

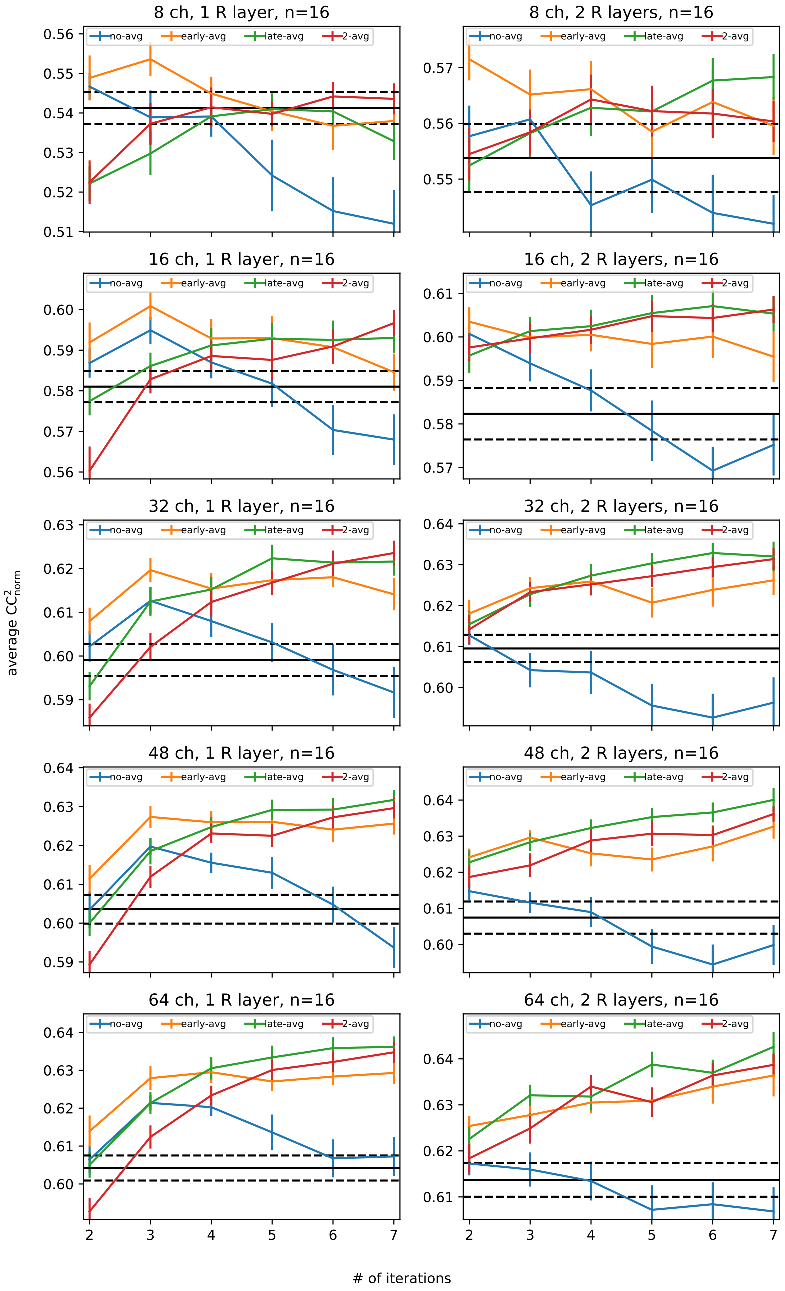


Figure B.7: Same as Figure B.6, using 50% of available training data.

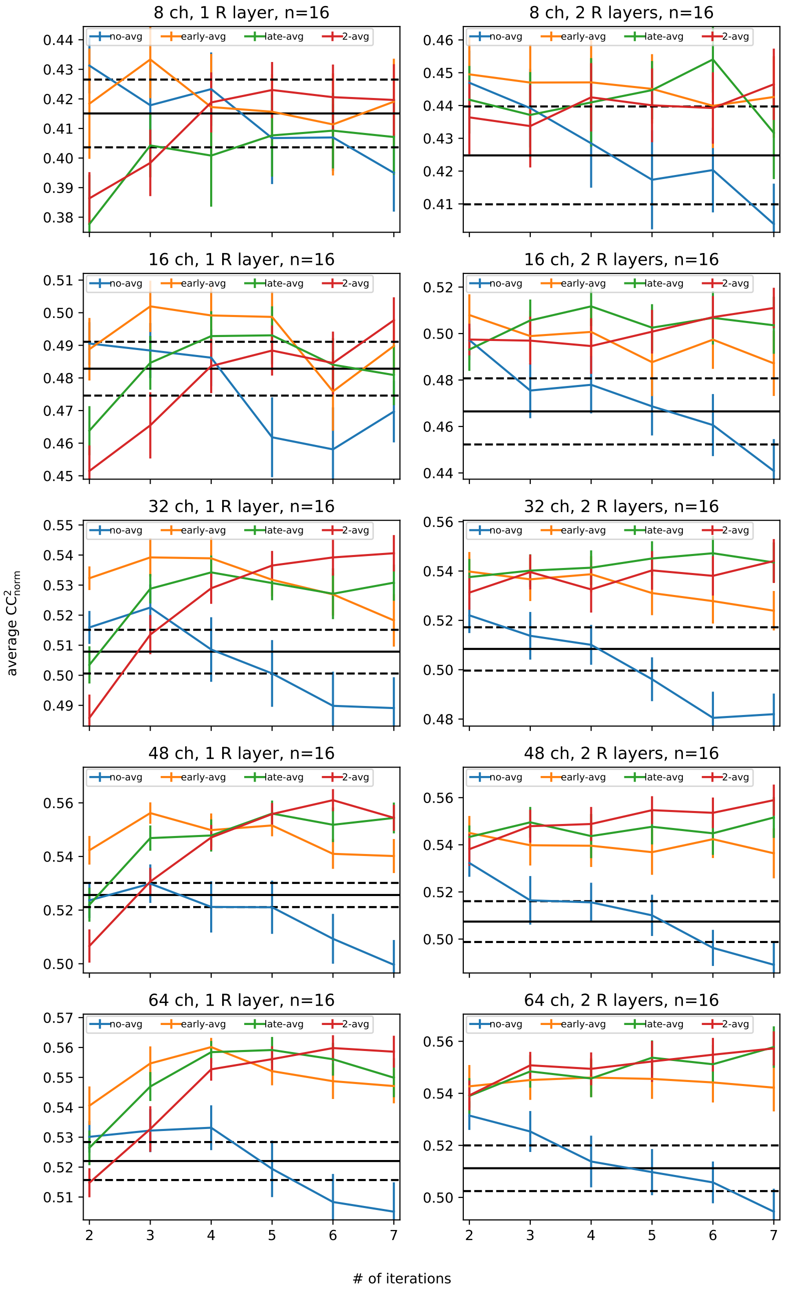


Figure B.8: Same as Figure B.6, using 25% of available training data.

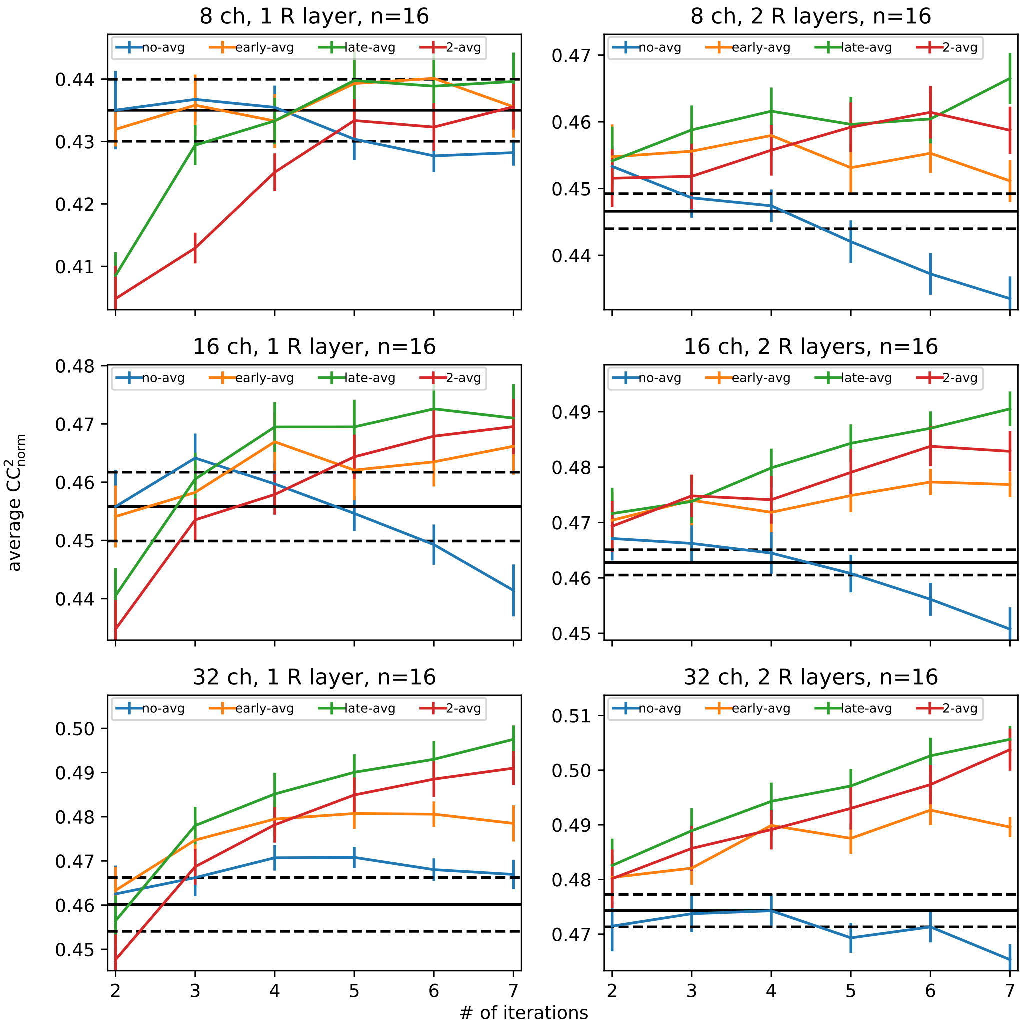


Figure B.9: The NS 2250 version of Figure B.6.

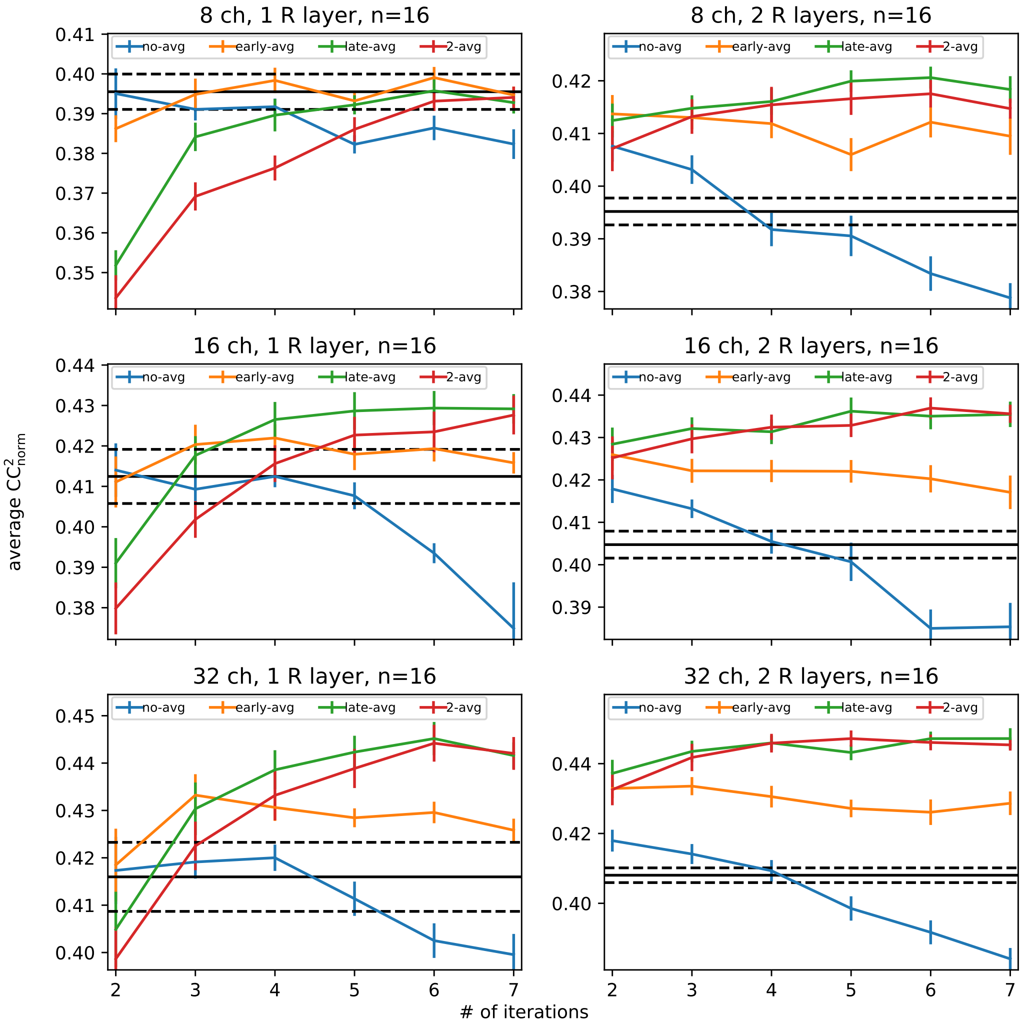


Figure B.10: The NS 2250 version of Figure B.7.

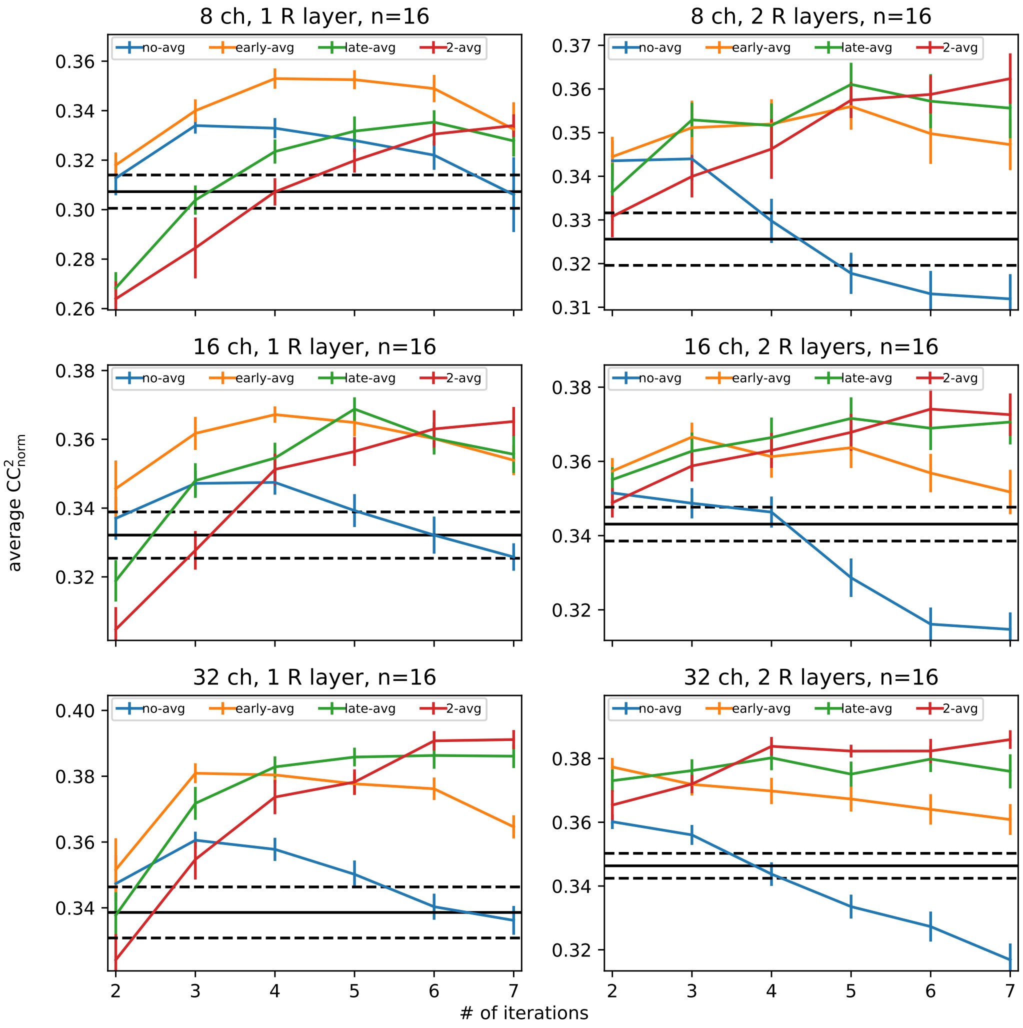
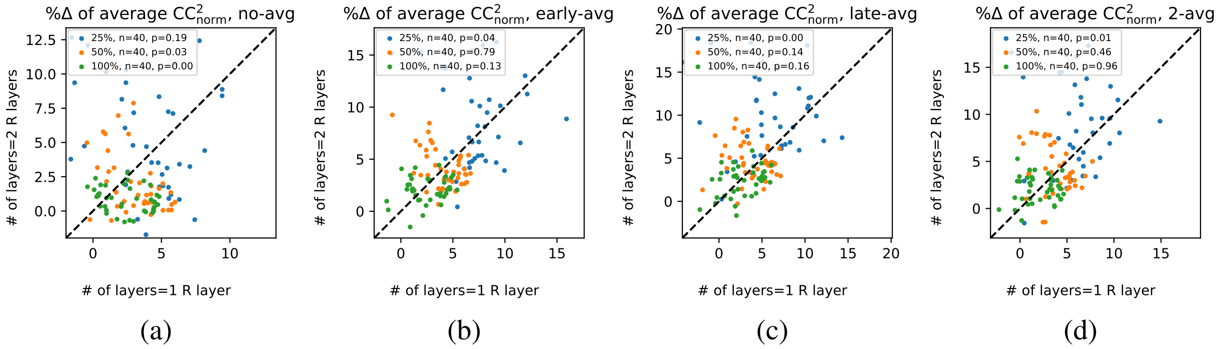


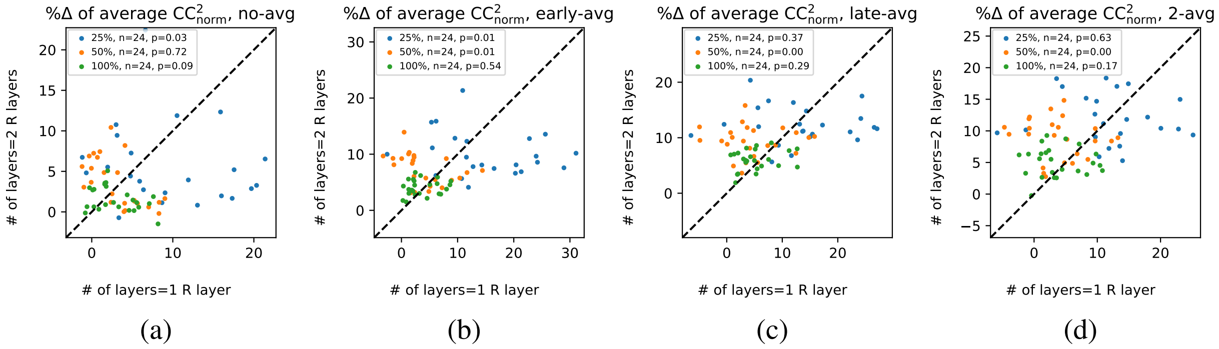
Figure B.11: The NS 2250 version of Figure B.8.

B.8.3 Overall results



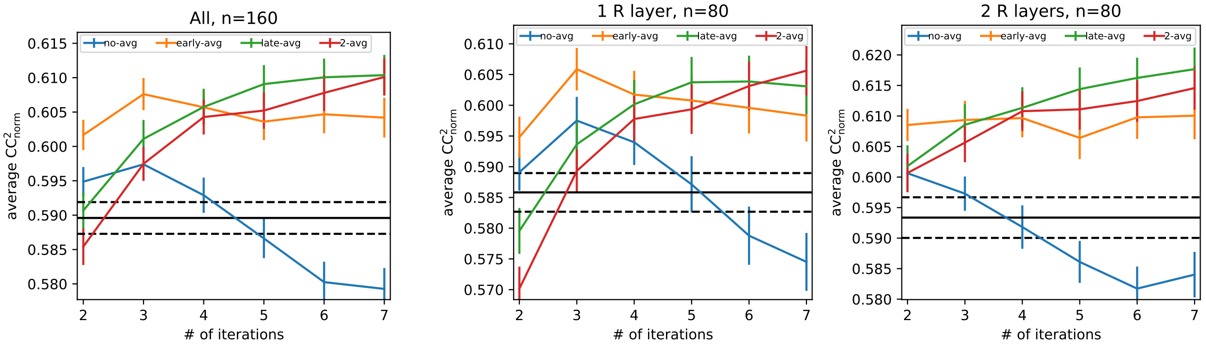
(a) (b) (c) (d)

Figure B.12: The relative performance metric changes of recurrent models over feed-forward models under different numbers of recurrent layers on the ImageNet 8K data set. The results are presented in the same style as those in Figure 4.8.



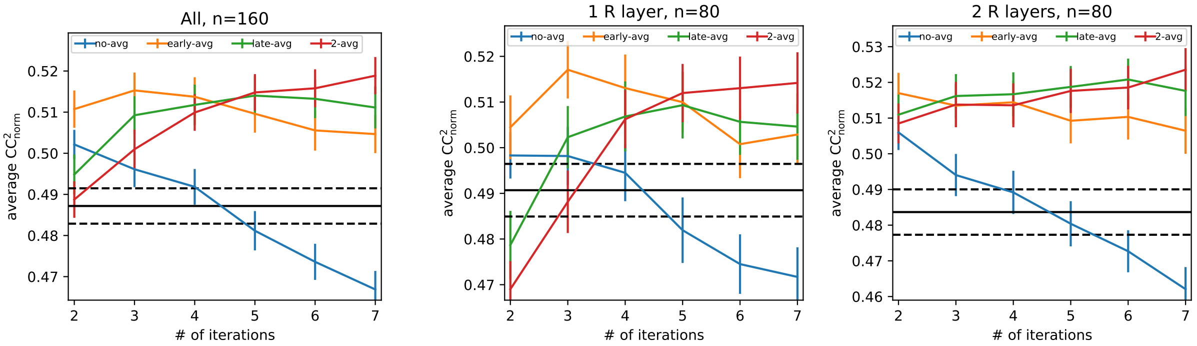
(a) (b) (c) (d)

Figure B.13: The NS 2250 version of Figure B.12.



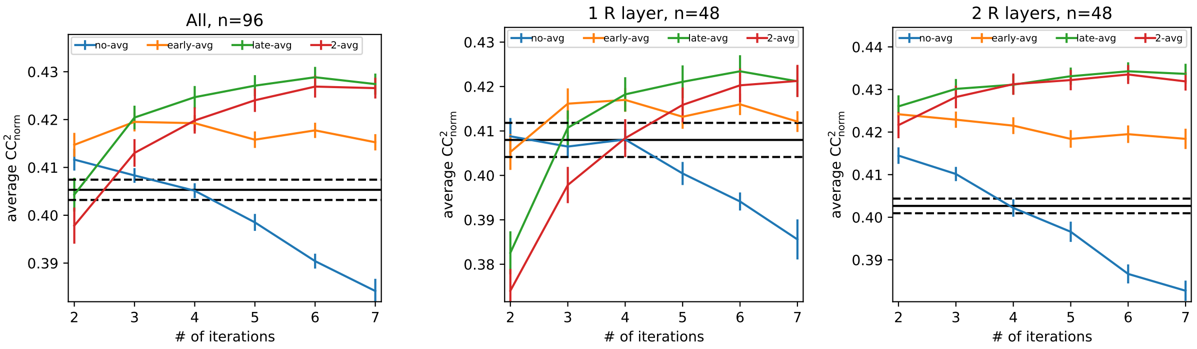
(a) (b)

Figure B.14: Same as Figure 4.11, with 50% of available training data.



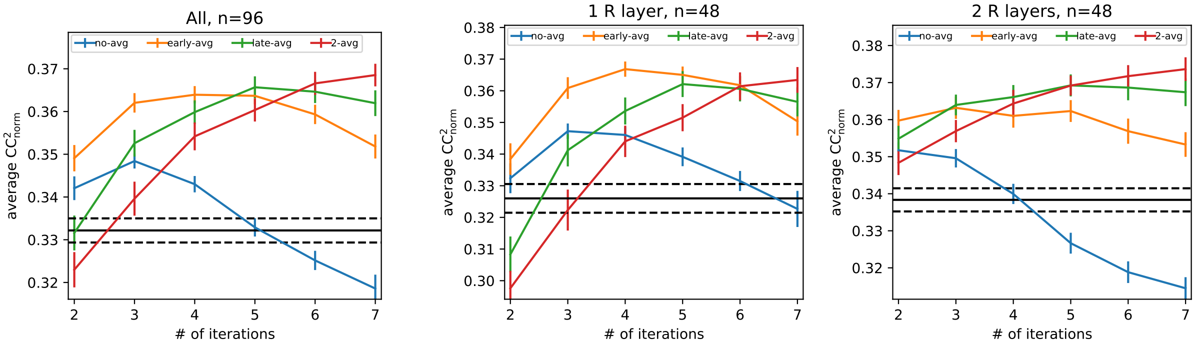
(a) (b)

Figure B.15: Same as Figure 4.11, with 25% of available training data.



(a) (b)

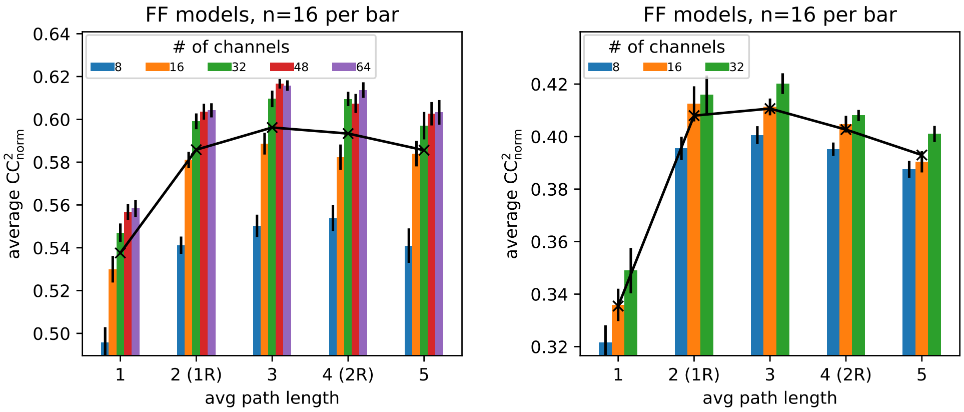
Figure B.16: Same as Figure 4.12, with 50% of available training data.



(a) (b)

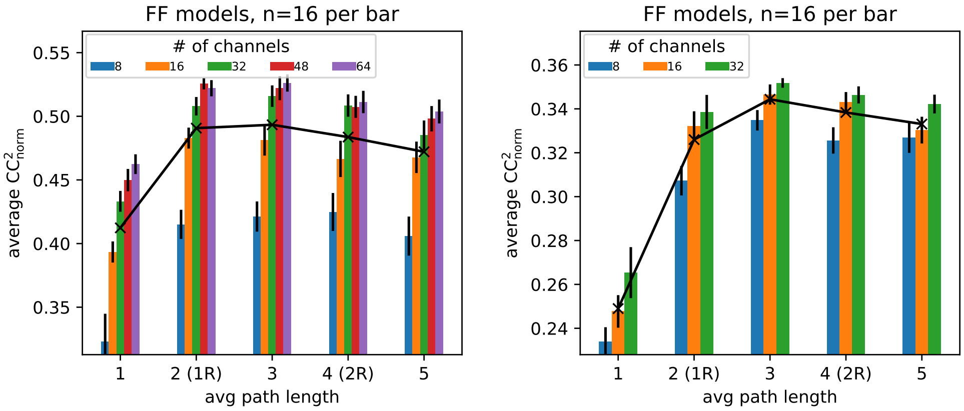
Figure B.17: Same as Figure 4.12, with 25% of available training data.

B.9 Additional results on path length and diversity



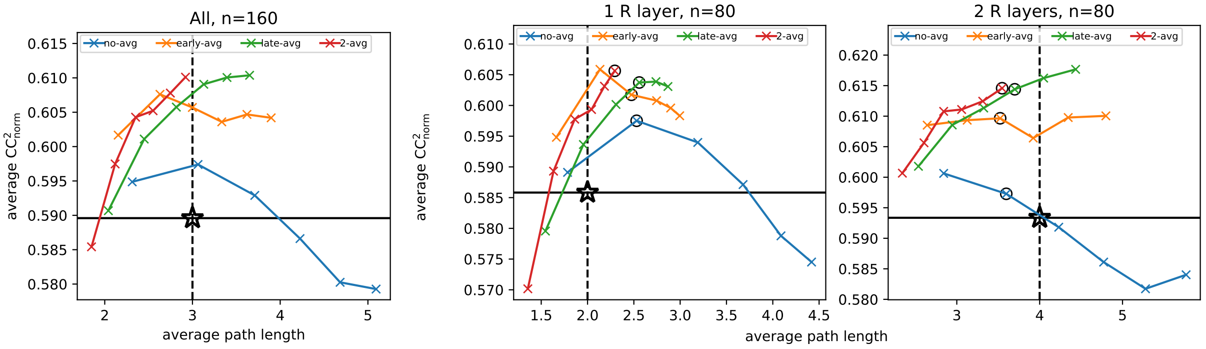
(a) (b)

Figure B.18: Same as Figure 4.19, for models trained using 50% of training data.



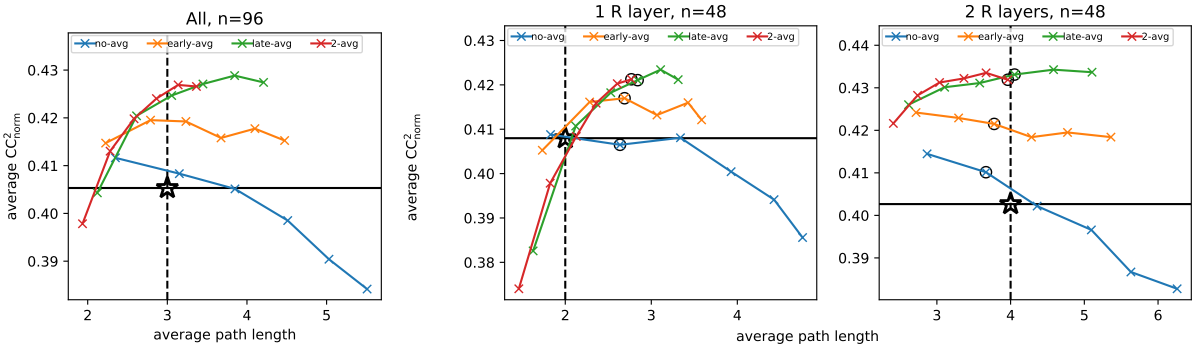
(a) (b)

Figure B.19: Same as Figure 4.19, for models trained using 25% of training data.



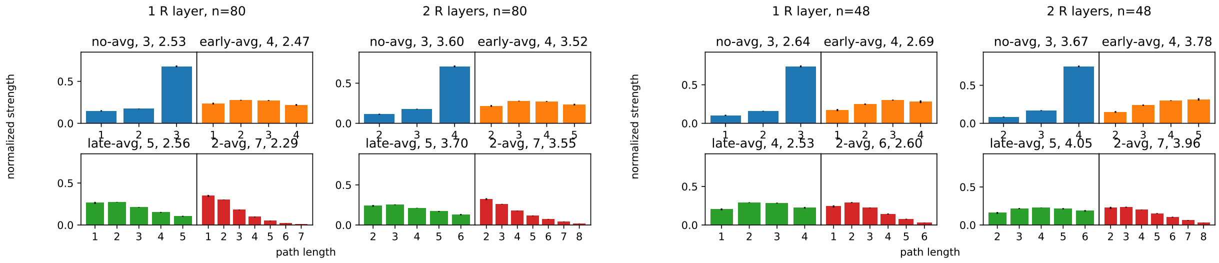
(a) (b)

Figure B.20: Same as Figure 4.17, with models trained using 50% of training data.



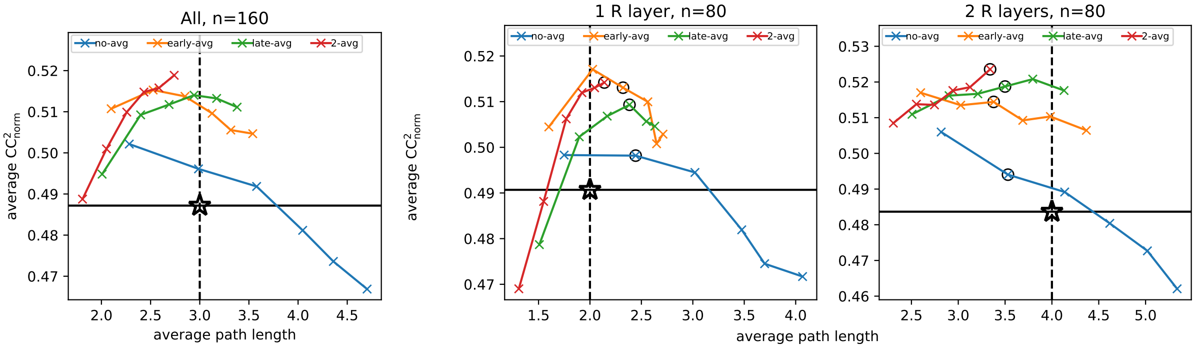
(a) (b)

Figure B.21: Same as Figure 4.18, with models trained using 50% of training data.



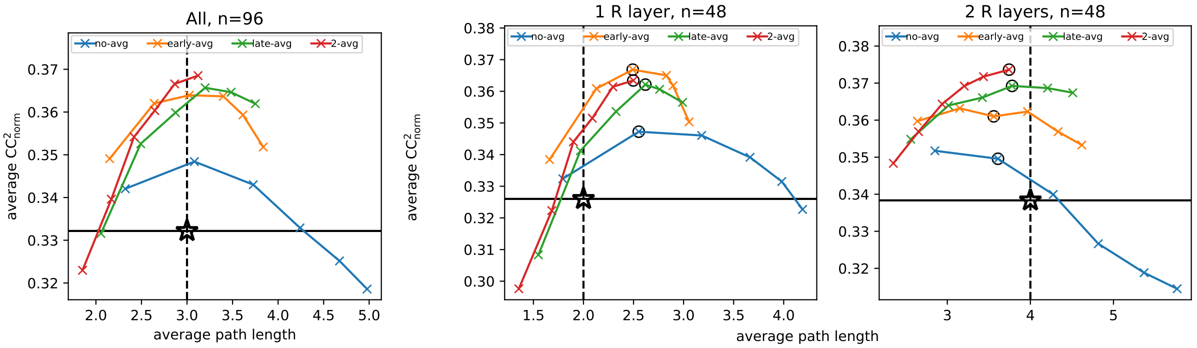
(a) (b)

Figure B.22: Same as Figure 4.20, with models trained using 50% of training data.



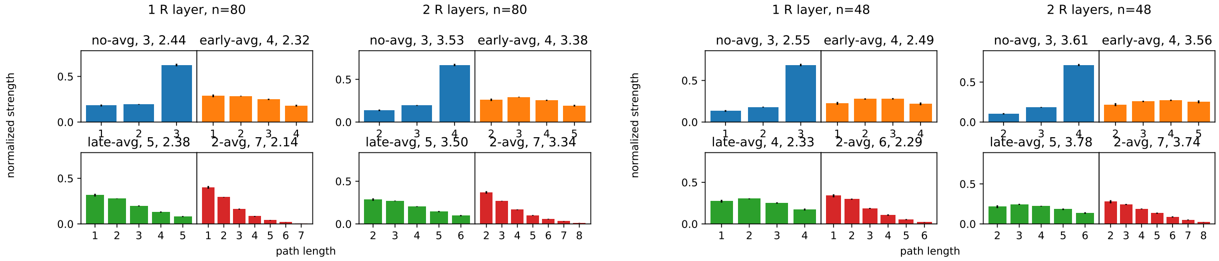
(a) (b)

Figure B.23: Same as Figure 4.17, with models trained using 25% of training data.



(a) (b)

Figure B.24: Same as Figure 4.18, with models trained using 25% of training data.



(a) (b)

Figure B.25: Same as Figure 4.20, with models trained using 25% of training data.

B.10 Additional results using a different random seed for splitting 8K data into training, validation, and testing sets

Due to resource limits, for each data set, we've presented all the results in this study using a single random seed for splitting data into training, validation, and testing sets (Section 4.3.3.2). As a sanity check, we trained additional models with a different random seed for data splitting, in order to make sure that our main results that recurrent models outperformed feed-forward ones would still hold regardless of data splitting. Due to resource limits we only retrained the subset of models using 100% of training data on ImageNet 8K; results on these retrained models were consistent with the results on the original models as shown below.

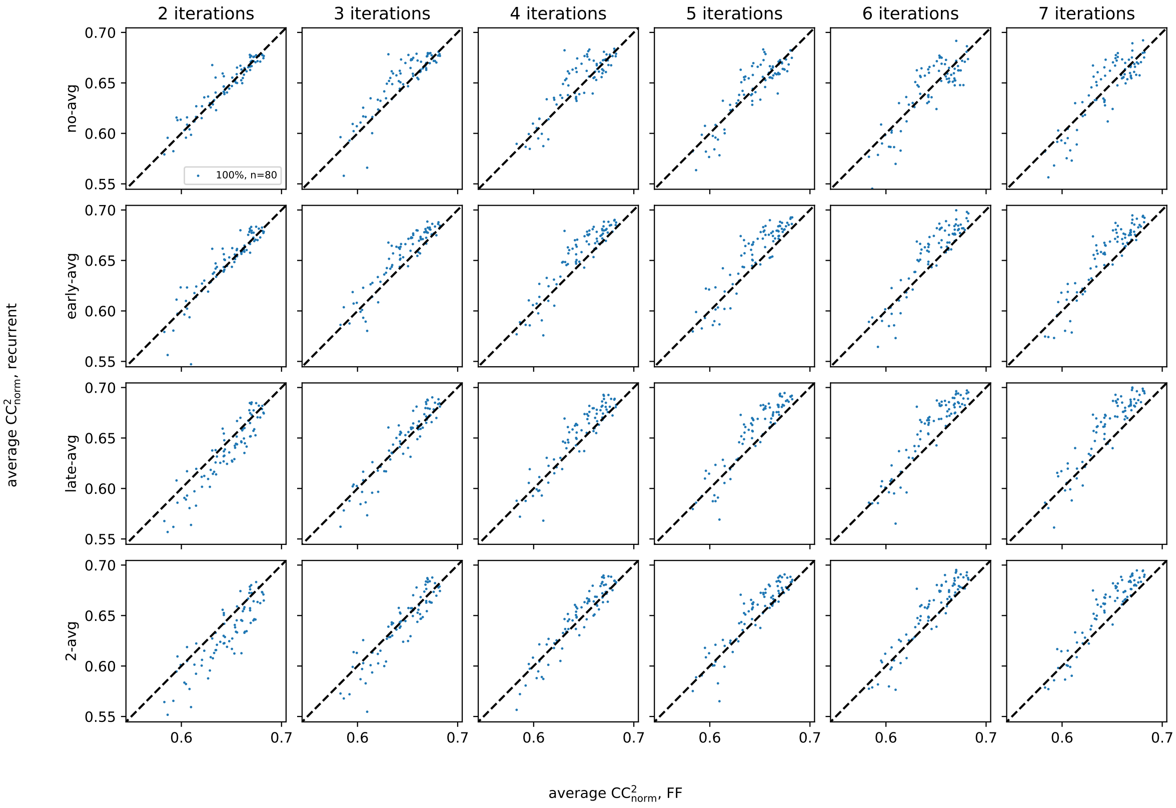


Figure B.26: Same as Figure 4.6, using another data splitting seed and only using 100% of training data.

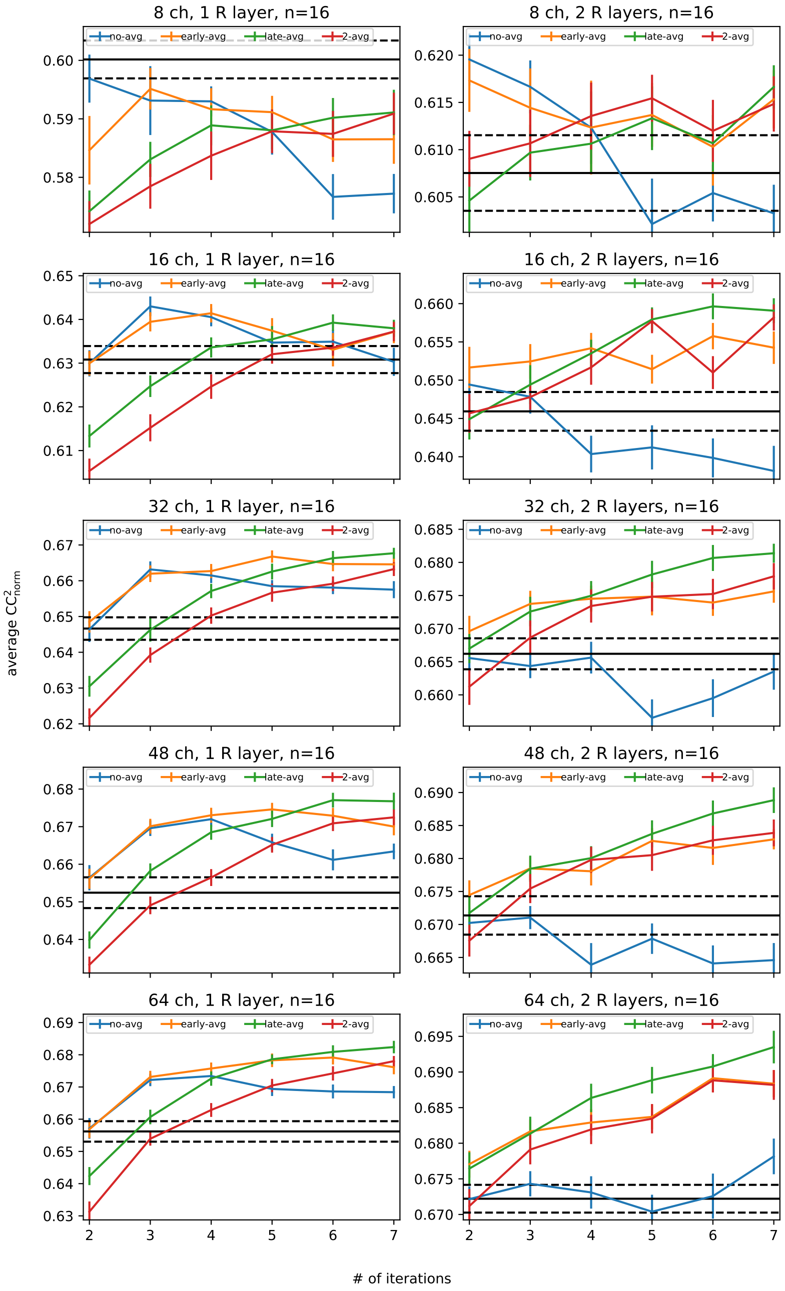
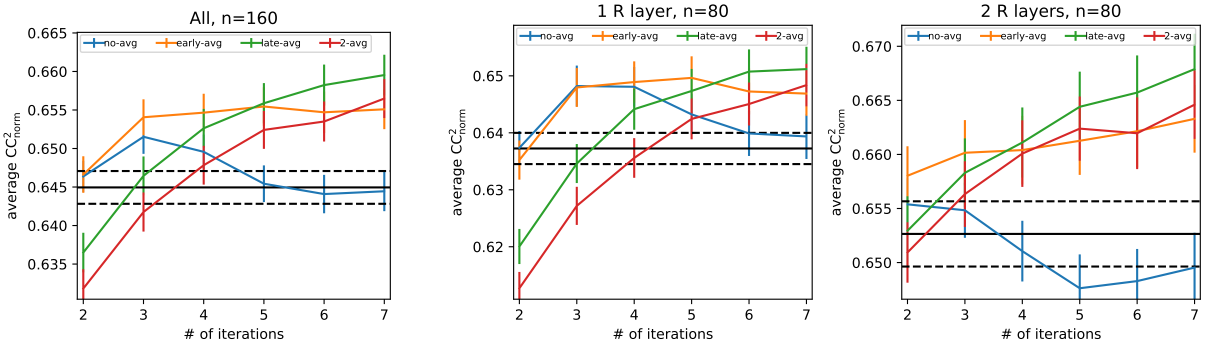


Figure B.27: Same as Figure B.6, with another data splitting seed.



(a) (b)

Figure B.28: Same as Figure 4.11, using another data splitting seed.

Bibliography

[1] Edward H. Adelson and James R. Bergen. Spatiotemporal energy models for the perception of motion. *J. Opt. Soc. Am. A*, 2(2):284–299, Feb 1985. doi: 10.1364/JOSAA.2.000284. URL http://josaa.osa.org/abstract.cfm?URI=josaa-2-2-284. 1.1, 3.1, 3.3.2.2

[2] A M Aertsen, G L Gerstein, M K Habib, and G Palm. Dynamics of neuronal firing correlation: modulation of ”effective connectivity”. *Journal of Neurophysiology*, 61(5):900–917, May 1989. 2.2.3.2, 2.4.1

[3] B W Andrews and D A Pollen. Relationship between spatial frequency selectivity and receptive field profile of simple cells. *The Journal of Physiology*, 287(1):163–176, 1979. ISSN 1469-7793. doi: 10.1113/jphysiol.1979.sp012652. URL http://dx.doi.org/10.1113/jphysiol.1979.sp012652. 3.6.1

[4] Akiyuki Anzai, Xinmiao Peng, and David C Van Essen. Neurons in monkey visual area v2 encode combinations of orientations. *Nature neuroscience*, 10(10):1313–1321, 2007. 1.1

[5] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 688–699, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html. 4.2.2, 4.5.2.4

[6] P N Belhumeur and D Mumford. A Bayesian treatment of the stereo correspondence problem using half-occluded regions. In *1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 506–512. IEEE Comput. Soc. Press, 1992. 2.2.2.1, 2.3.2

[7] Christopher M Bishop. *Machine learning and pattern recognition*. Information Science and Statistics. Springer, 2006. URL https://www.microsoft.com/en-us/research/people/cmbishop/. 3.3.4, 4.5.1

[8] William H Bosking, Justin C Crowley, and David Fitzpatrick. Spatial coding of position and orientation in primary visual cortex. *Nature Neuroscience*, 5(9):874–882, August 2002. 2.1

[9] C. E. Bredfeldt and D. L. Ringach. Dynamics of spatial frequency tuning in macaque

v1. *Journal of Neuroscience*, 22(5):1976–1984, 2002. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.22-05-01976.2002. URL `https://www.jneurosci.org/content/22/5/1976`. 4.5.2.1, 4.5.2.2, 4.5.3.3

[10] Carlos D Brody. Disambiguating Different Covariation Types. *Neural Computation*, 11 (7):1527–1535, October 1999. 2.2.3.2

[11] Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. Neural Dynamics as Sampling: A Model for Stochastic Computation in Recurrent Networks of Spiking Neurons. *PLoS Comput Biol*, 7(11):e1002211, November 2011. 2.4.2

[12] Max F. Burg, Santiago A. Cadena, George H. Denfield, Edgar Y. Walker, Andreas S. Tolias, Matthias Bethge, and Alexander S. Ecker. Learning divisive normalization in primary visual cortex. *bioRxiv*, 2020. doi: 10.1101/767285. URL `https://www.biorxiv.org/content/early/2020/07/24/767285`. 4.5.2.2

[13] 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):1–27, 04 2019. doi: 10.1371/journal.pcbi.1006897. URL `https://doi.org/10.1371/journal.pcbi.1006897`. (document), 1.1, 1.2.3, 3.1, 3.2.1.0.1, 3.3.5, 3.5.4, 3.6.2, 3.6.3, 4.1, 4.2.1, 4.3.1.1, 4.3.2, 4.3.2.1, 4.4, 4.3.3.1, 4.5.2.2, 4.5.2.4, A.1.2, A.1.3, A.1.5, B.3.1.1

[14] Matteo Carandini and David J Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51–62, January 2012. 2.4.3

[15] Matteo Carandini, Jonathan B Demb, Valerio Mante, David J Tolhurst, Yang Dan, Bruno A Olshausen, Jack L Gallant, and Nicole C Rust. Do We Know What the Early Visual System Does? *Journal of Neuroscience*, 25(46):10577–10597, November 2005. doi: 10.1523/JNEUROSCI.3726-05.2005. URL `http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.3726-05.2005`. 3.1, 3.3.2.1, 3.3, 3.3.2.2, 3.3.2.3, 3.3.3.1, 3.3.3.2, 4.3.1.1, 4.5.2.2

[16] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html`. 4.2.2, 4.5.2.3

[17] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`. 4.5.2.4

[18] Ruben Coen-Cagli, Peter Dayan, and Odelia Schwartz. Statistical Models of Linear and Nonlinear Contextual Interactions in Early Visual Processing. In Yoshua Bengio, Dale Schuurmans, John D Lafferty, Christopher K I Williams, and Aron

Culotta, editors, *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 369–377. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3794-statistical-models-of-linear-and-nonlinear-contextual-interaction 1.1, 4.2.1

[19] Ruben Coen-Cagli, Peter Dayan, and Odelia Schwartz. Cortical Surround Interactions and Perceptual Salience via Natural Scene Statistics. *PLoS computational biology*, 8(3): e1002405, March 2012. doi: 10.1371/journal.pcbi.1002405. URL http://dx.plos.org/10.1371/journal.pcbi.1002405. 1.1, 4.2.1

[20] Ruben Coen-Cagli, Adam Kohn, and Odelia Schwartz. Flexible gating of contextual influences in natural vision. *Nature Neuroscience*, 18(11):1648–1655, October 2015. doi: 10.1038/nn.4128. URL http://www.nature.com/articles/nn.4128. 1.1, 3.2.1.0.1, 4.2.1, 4.3.2, A.1.5

[21] Marlene R Cohen and John H R Maunsell. Attention improves performance primarily by reducing interneuronal correlations. *Nature Neuroscience*, 12(12):1594–1600, December 2009. 2.3

[22] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=BydARw9ex. 4.5.1, 4.5.2.4, 4.5.3.3, 4.5.3.4

[23] Rui Ponte Costa, Ioannis Alexandros M. Assael, Brendan Shillingford, Nando de Freitas, and Tim P. Vogels. Cortical microcircuits as gated-recurrent neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 271–282, 2017. URL http://papers.nips.cc/paper/6631-cortical-microcircuits-as-gated-recurrent-neural-networks. 1.1

[24] John G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *J. Opt. Soc. Am. A*, 2(7): 1160–1169, Jul 1985. doi: 10.1364/JOSAA.2.001160. URL http://josaa.osa.org/abstract.cfm?URI=josaa-2-7-1160. 3.3.2

[25] Stephen V David and Jack L Gallant. Predicting neuronal responses during natural vision. *Network: Computation in Neural Systems*, 16(2-3):239–260, 2005. doi: 10.1080/09548980500464030. URL http://www.tandfonline.com/doi/full/10.1080/09548980500464030. 1.1, 3.1, 3.2.1.0.1, 3.3.3.1, 3.3.3.2, 3.4.3.1, 4.5.2.2, A.1.5

[26] Peter Dayan and L F Abbott. *Theoretical Neuroscience*. Computa-

tional and Mathematical Modeling of Neural Systems. MIT Press, 2001. ISBN 0262541858. URL `http://www.worldcat.org/title/theoretical-neuroscience-computational-and-mathematical-modeling-of-ne` `oclc/952504127`. 3.1, 3.3.2

[27] Rodney J. Douglas and Kevan A. C. Martin. Recurrent neuronal circuits in the neocortex. *Current Biology*, 17(13):R496–R500, 2018/10/18 2007. doi: 10.1016/j.cub.2007.04.024. URL `https://doi.org/10.1016/j.cub.2007.04.024`. 1.1

[28] A S Ecker, P Berens, G A Keliris, M Bethge, N K Logothetis, and A S Tolias. Decorrelated Neuronal Firing in Cortical Microcircuits. *Science*, 327(5965):584–587, January 2010. 2.3

[29] 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. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=H1fU8iAqKX`. 4.5.2.2

[30] B Efron and Robert John Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994. 2.2.3.2

[31] James H Elder and Richard M Goldberg. Ecological statistics of Gestalt laws for the perceptual organization of contours. *Journal of Vision*, 2(4):324–353, June 2002. 2.1, 2.4.1

[32] Daniel J. Felleman and David C. Van Essen. Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral Cortex*, 1(1):1–47, 01 1991. ISSN 1047-3211. doi: 10.1093/cercor/1.1.1-a. URL `https://doi.org/10.1093/cercor/1.1.1-a`. 1.1, 1.2.3, 4.1, 4.2.1, 4.5.1

[33] David J Field, Anthony Hayes, and Robert F Hess. Contour integration by the human visual system: Evidence for a local “association field”. *Vision Research*, 33(2):173–193, January 1993. (document), 2.1, 2.3.2, 2.5, 2.4.1

[34] Ian M. Finn and David Ferster. Computational diversity in complex cells of cat primary visual cortex. *Journal of Neuroscience*, 27(36):9638–9648, 2007. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.2119-07.2007. URL `http://www.jneurosci.org/content/27/36/9638`. 3.6.1

[35] József Fiser, Pietro Berkes, Gergő Orbán, and Máté Lengyel. Statistically optimal perception and learning: from behavior to neural representations. *Trends in Cognitive Sciences*, 14(3):119–130, March 2010. 2.4.2

[36] David J Fleet, Hermann Wagner, and David J Heeger. Neural encoding of binocular disparity: Energy models, position shifts and phase shifts. *Vision Research*, 36(12):1839–1857, June 1996. 2.3

[37] K H Foster, J P Gaska, M Nagler, and D A Pollen. Spatial and temporal frequency selectivity of neurones in visual cortical areas v1 and v2 of the macaque mon-

key. *The Journal of Physiology*, 365(1):331–363, 1985. doi: 10.1113/jphysiol.1985.sp015776. URL https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1985.sp015776. 1.1

[38] J Friedman, T Hastie, and R Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, June 2008. 2.4.1

[39] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software, Articles*, 33(1):1–22, 2010. ISSN 1548-7660. doi: 10.18637/jss.v033.i01. URL https://www.jstatsoft.org/v033/i01. 3.4.3.3

[40] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980. ISSN 1432-0770. doi: 10.1007/BF00344251. URL https://doi.org/10.1007/BF00344251. 3.6.1

[41] Deep Ganguli and Eero P Simoncelli. Implicit encoding of prior probabilities in optimal neural populations. In J D Lafferty, C K I Williams, J Shawe-Taylor, R S Zemel, and A Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 658–666. Curran Associates, Inc., 2010. (document), 2.2.2.1.1, 2.2, 2.4, 2.4.2

[42] W S Geisler, J S Perry, B J Super, and D P Gallogly. Edge co-occurrence in natural images predicts contour grouping performance. *Vision Research*, 41(6):711–724, March 2001. 2.1, 2.4.1

[43] Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, November 1984. 2.2.2.1

[44] George L Gerstein and Kyle L Kirkland. Neural assemblies: technical issues, analysis, and modeling. *Neural Networks*, 14(6-7):589–598, July 2001. 2.2.3.2

[45] Tim Gollisch and Markus Meister. Eye smarter than scientists believed: Neural computations in circuits of the retina. *Neuron*, 65(2):150–164, jan 2010. doi: 10.1016/j.neuron.2009.12.009. URL http://dx.doi.org/10.1016/j.neuron.2009.12.009. 3.5.2.0.1

[46] Ian J Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 3.4.1.2, 4.5.2.4, A.1.4

[47] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015. URL http://arxiv.org/abs/1503.04069. 4.5.1, 4.5.2.4, 4.5.3.4

[48] Kuan Han, Haiguang Wen, Yizhen Zhang, Di Fu, Eugenio Culurciello, and Zhongming Liu. Deep predictive coding network with local recurrent processing for object recognition. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal,*

Canada., pages 9221–9233, 2018. URL http://papers.nips.cc/paper/8133-deep-predictive-coding-network-with-local-recurrent-processing-fo 4.5.1, 4.5.2.1, 4.5.3.1, 4.5.3.4

[49] H. K. Hartline. The response of single optic nerve fibers of the vertebrate eye to illumination of the retina. *American Journal of Physiology-Legacy Content*, 121(2):400–415, 1938. doi: 10.1152/ajplegacy.1938.121.2.400. URL https://doi.org/10.1152/ajplegacy.1938.121.2.400. 1.1

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90. 4.2.2

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016. doi: 10.1007/978-3-319-46493-0_38. URL https://doi.org/10.1007/978-3-319-46493-0_38. 4.2.2

[52] David J. Heeger. Half-squaring in responses of cat striate cells. *Visual Neuroscience*, 9 (5):427–443, 1992. doi: 10.1017/S095252380001124X. 1.1, 3.1, 3.3.2.1, 3.3.2.1, 3.3.2.2, 3.5.2.0.2, 3.6.1

[53] Jay Hegdé and David C. Van Essen. Selectivity for complex shapes in primate visual area v2. *Journal of Neuroscience*, 20(5):RC61–RC61, 2000. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.20-05-j0001.2000. URL https://www.jneurosci.org/content/20/5/RC61. 1.1

[54] Jay Hegdé and David C Van Essen. A comparative study of shape representation in macaque visual areas V2 and V4. *Cerebral Cortex*, 17(5):1100–1116, 2007. 1.1, 3.1

[55] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. URL https://science.sciencemag.org/content/313/5786/504. 4.2.2, 4.5.2.4

[56] Geoffrey E Hinton and T J Sejnowski. Learning and Relearning in Boltzmann Machines. In *Parallel Distributed Processing: Foundations*. MIT Press, 1986. 2.1, 2.2.2.1

[57] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006. doi: 10.1162/neco.2006.18.7.1527. URL https://doi.org/10.1162/neco.2006.18.7.1527. 4.2.2, 4.5.2.4

[58] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. 4.5.2.4

[59] Patrik O Hoyer and Aapo Hyvärinen. Interpreting Neural Response Variability as Monte Carlo Sampling of the Posterior. In S Becker, S Thrun, and K Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 293–300. MIT Press, 2003. 2.4.2

[60] Anne Hsu, Alexander Borst, and Frédéric Theunissen. Quantifying variability in neural responses and its application for the validation of model predictions. *Network: Computation in Neural Systems*, 15(2):91–109, May 2004. doi: 10.1088/0954-898X/15/2/002. URL `http://www.informaworld.com/openurl?genre=article&doi=10.1088/0954-898X/15/2/002&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3`. 3.3.6, 3.3.6, 4.3.3.4, B.4

[61] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.243. URL `https://doi.org/10.1109/CVPR.2017.243`. 4.2.2

[62] Jinggang Huang, A B Lee, and D Mumford. Statistics of range images. In *2000 IEEE Conference on Computer Vision and Pattern Recognition*, pages 324–331. IEEE Comput. Soc, 2000. 2.2.1

[63] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959. ISSN 1469-7793. doi: 10.1113/jphysiol.1959.sp006308. URL `http://dx.doi.org/10.1113/jphysiol.1959.sp006308`. 1.1, 3.1, 3.3.3.1

[64] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962. ISSN 1469-7793. doi: 10.1113/jphysiol.1962.sp006837. URL `http://dx.doi.org/10.1113/jphysiol.1962.sp006837`. 1.1, 3.1, 3.6.1

[65] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968. ISSN 1469-7793. doi: 10.1113/jphysiol.1968.sp008455. URL `http://dx.doi.org/10.1113/jphysiol.1968.sp008455`. 1.1, 3.1

[66] David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of Neurophysiology*, 28(2):229–289, 1965. doi: 10.1152/jn.1965.28.2.229. URL `https://doi.org/10.1152/jn.1965.28.2.229`. PMID: 14283058. 1.1

[67] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL `http://jmlr.org/proceedings/papers/v37/ioffe15.html`. 4.3.1.1

[68] Minami Ito and Hidehiko Komatsu. Representation of angles embedded within contour

stimuli in area v2 of macaque monkeys. *Journal of Neuroscience*, 24(13):3313–3324, 2004. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.4364-03.2004. URL `https://www.jneurosci.org/content/24/13/3313`. 1.1

[69] J P Jones and L A Palmer. The two-dimensional spatial structure of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58(6):1187–1211, 1987. URL `http://jn.physiology.org/content/58/6/1187`. 3.3.3.1

[70] J P Jones and L A Palmer. An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58(6):1233–1258, 1987. URL `http://jn.physiology.org/content/58/6/1233`. 3.1, 3.3.2, 3.5.1.0.2

[71] Mitesh K Kapadia, Minami Ito, Charles D Gilbert, and Gerald Westheimer. Improvement in visual sensitivity by changes in local context: Parallel studies in human observers and in V1 of alert monkeys. *Neuron*, 15(4):843–856, October 1995. 2.1

[72] Mitesh K. Kapadia, Gerald Westheimer, and Charles D. Gilbert. Dynamics of spatial summation in primary visual cortex of alert monkeys. *Proceedings of the National Academy of Sciences*, 96(21):12073–12078, 1999. ISSN 0027-8424. doi: 10.1073/pnas.96.21.12073. URL `https://www.pnas.org/content/96/21/12073`. 4.5.2.1, 4.5.2.2, 4.5.3.3

[73] Mitesh K. Kapadia, Gerald Westheimer, and Charles D. Gilbert. Spatial distribution of contextual interactions in primary visual cortex and in visual perception. *Journal of Neurophysiology*, 84(4):2048–2062, 2000. doi: 10.1152/jn.2000.84.4.2048. URL `https://doi.org/10.1152/jn.2000.84.4.2048`. PMID: 11024097. 1.1, 4.2.1

[74] Ryan C. Kelly, Matthew A. Smith, Jason M. Samonds, Adam Kohn, A. B. Bonds, J. Anthony Movshon, and Tai Sing Lee. Comparison of recordings from microelectrode arrays and single electrodes in the visual cortex. *Journal of Neuroscience*, 27 (2):261–264, 2007. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.4906-06.2007. URL `https://www.jneurosci.org/content/27/2/261`. B.1.1

[75] Ryan C. Kelly, Matthew A. Smith, Robert E. Kass, and Tai Sing Lee. Accounting for network effects in neuronal responses using L1 regularized point process models. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pages 1099–1107. Curran Associates, Inc., 2010. URL `http://papers.nips.cc/paper/4050-accounting-for-network-effects-in-neuronal-responses-using-l1-reg` 3.1, 4.5.2.2

[76] Ryan C Kelly, Matthew A Smith, Robert E Kass, and Tai Sing Lee. Local field potentials indicate network state and account for neuronal response variability. *Journal of Computational Neuroscience*, 29(3):567–579, January 2010. 2.3

[77] Tim C. Kietzmann, Courtney J. Spoerer, Lynn K. A. Sörensen, Radoslaw M. Cichy, Olaf

Hauk, and Nikolaus Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43):21854–21863, 2019. ISSN 0027-8424. doi: 10.1073/pnas.1905544116. URL https://www.pnas.org/content/116/43/21854. 1.2.3, 4.1, 4.2.1, 4.5.2.1

[78] William F. Kindel, Elijah D. Christensen, and Joel Zylberberg. Using deep learning to probe the neural code for images in primary visual cortex. *Journal of Vision*, 19(4):29–29, 04 2019. ISSN 1534-7362. doi: 10.1167/19.4.29. URL https://doi.org/10.1167/19.4.29. 1.1, 1.2.3, 3.1, 3.2.1.0.1, 3.3.5, 3.6.3, 4.1, 4.2.1, A.1.3, A.1.5, B.3.1.1, B.3.1.2

[79] P D King, J Zylberberg, and M R DeWeese. Inhibitory Interneurons Decorrelate Excitatory Cells to Drive Sparse Code Formation in a Spiking Model of V1. *Journal of Neuroscience*, 33(13):5475–5485, March 2013. doi: 10.1523/JNEUROSCI.4188-12.2013. URL http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.4188-12.2013. 2.4.2, 4.5.2.2

[80] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980. 3.3, 3.4.2

[81] David Klindt, Alexander S. Ecker, Thomas Euler, and Matthias Bethge. Neural system identification for large populations separating ”what” and ”where”. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3509–3519, 2017. URL http://papers.nips.cc/paper/6942-neural-system-identification-for-large-populations-separating-wha 1.1, 1.2.3, 3.6.3, 4.1, 4.2.1, 4.3.1.1, 4.3.3.1, 4.3.3.3, 4.3.3.3.1, 4.3.3.3.2, 4.5.2.2, 4.5.2.4, A.1.2, A.1.3, B.2.1

[82] David C Knill and Whitman Richards. *Perception as Bayesian inference*. Cambridge University Press, September 1996. 2.1

[83] Ho Ko, Sonja B Hofer, Bruno Pichler, Katherine A Buchanan, P Jesper Sjöström, and Thomas D Mrsic-Flogel. Functional specificity of local synaptic connections in neocortical networks. *Nature*, 473(7345):87–91, April 2011. 2.4.1

[84] Ho Ko, Lee Cossell, Chiara Baragli, Jan Antolik, Claudia Clopath, Sonja B Hofer, and Thomas D Mrsic-Flogel. The emergence of functional microcircuits in visual cortex. *Nature*, 496(7443):96–100, April 2013. 2.4.1

[85] C Koch, J Marroquin, and A Yuille. Analog ”neuronal” networks in early vision. *Proceedings of the National Academy of Sciences*, 83(12):4263–4267, June 1986. 2.2.2.1

[86] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009. 2.2.2.2.1, 2

[87] Urs Köster and Bruno Olshausen. Testing our conceptual understanding of V1 function. *ArXiv e-prints*, q-bio.NC, November 2013. URL https://arxiv.org/abs/1311.0778. 1.1, 3.1, 3.3.3.2

[88] Raghavendra Kotikalapudi. keras-vis: Keras visualization toolkit, 2017. https://github.com/raghakot/keras-vis. (document), 3.5.4, 3.7, 3.11

[89] Nikolaus Kriegeskorte. Deep neural networks: A new framework for modeling biological vision and brain information processing. *Annual Review of Vision Science*, 1(1):417–446, 2015. doi: 10.1146/annurev-vision-082114-035447. URL `https://doi.org/10.1146/annurev-vision-082114-035447`. PMID: 28532370. 1.1, 3.1, 3.3.5, 3.6.2, 4.1, 4.2.1

[90] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational similarity analysis - connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience*, 2:4, 2008. ISSN 1662-5137. doi: 10.3389/neuro.06.004.2008. URL `https://www.frontiersin.org/article/10.3389/neuro.06.004.2008`. 1

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks`. 3.3.1

[92] Norbert Krüger. Collinearity and Parallelism are Statistically Significant Second-Order Relations of Complex Cell Responses. *Neural Processing Letters*, 8(2):117–129, 1998. 2.4.1

[93] Jonas Kubilius, Martin Schrimpf, Ha Hong, Najib J. Majaj, Rishi Rajalingham, Elias B. Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Kailyn Schmidt, Aran Nayebi, Daniel Bear, Daniel L. Yamins, and James J. DiCarlo. Brain-like object recognition with high-performing shallow recurrent anns. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12785–12796, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/7813d1590d28a7dd372ad54b5d29d033-Abstract.html`. 4.2.1, 4.5.1, 4.5.2.1, 4.5.3.1, 4.5.3.3, 4.5.3.4

[94] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=S1VaB4cex`. 4.2.2

[95] Brian Lau, Garrett B. Stanley, and Yang Dan. Computational subunits of visual cortical neurons revealed by artificial neural networks. *Proceedings of the National Academy of Sciences*, 99(13):8974–8979, 2002. doi: 10.1073/pnas.122173799. URL `http://www.`

pnas.org/content/99/13/8974.abstract. 4.5.2.2

[96] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area V2. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 873–880. Curran Associates, Inc., 2007. URL http://papers.nips.cc/paper/3313-sparse-deep-belief-net-model-for-visual-area-v2. 1.1, 4.5.2.2

[97] Tai Sing Lee. Computations in the early visual cortex. *Journal of Physiology-Paris*, 97(2):121–139, 2003. ISSN 0928-4257. doi: https://doi.org/10.1016/j.jphysparis.2003.09.015. URL https://www.sciencedirect.com/science/article/pii/S0928425703000573. Neurogeometry and visual perception. 1.1

[98] Tai Sing Lee. The Visual System's Internal Model of the World. *Proceedings of the IEEE*, 103(8):1359–1378, August 2015. 2.4.1

[99] Tai Sing Lee and David Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America A*, 20(7):1434–1448, 2003. 2.4.1

[100] Tai Sing Lee and Alan L Yuille. Efficient coding of visual scenes by grouping and segmentation. In *Bayesian Brain: Probabilistic Approaches to Neural Coding*, pages 141–185. MIT Press, 2006. 2.4.1

[101] Gang Li and S W Zucker. Differential Geometric Inference in Surface Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):72–86, January 2010. 2.3.2, 2.4.2

[102] Ming Li, Fang Liu, Hongfei Jiang, Tai Sing Lee, and Shiming Tang. Long-term two-photon imaging in awake macaque monkey. *Neuron*, 93(5):1049–1057.e3, Mar 2017. ISSN 0896-6273. doi: 10.1016/j.neuron.2017.01.027. URL http://dx.doi.org/10.1016/j.neuron.2017.01.027. 3.1, 3.2.2.0.2

[103] Wu Li and Charles D Gilbert. Global Contour Saliency and Local Colinear Interactions. *Journal of Neurophysiology*, 88(5):2846–2856, November 2002. 2.1

[104] Qianli Liao and Tomaso A. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *CoRR*, abs/1604.03640, 2016. URL http://arxiv.org/abs/1604.03640. 4.2.2, 4.5.2.3

[105] Grace W. Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, 0(0):1–15, 0. doi: 10.1162/jocn_a_01544. URL https://doi.org/10.1162/jocn_a_01544. PMID: 32027584. 1.1, 4.5.3.3

[106] Drew Linsley, Junkyung Kim, Vijay Veerabadran, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated-recurrent units. *CoRR*, abs/1805.08315, 2018. URL http://arxiv.org/abs/1805.08315. 1.1

[107] Yang Liu, Alan Conrad Bovik, and Lawrence K Cormack. Disparity statistics in natural

scenes. *Journal of Vision*, 8(11):19–19, August 2008. (document), 2.2.1, 2.1, 2.2.1.1, 2.2.1.1

[108] William Lotter, Gabriel Kreiman, and David D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *CoRR*, abs/1605.08104, 2016. URL `http://arxiv.org/abs/1605.08104`. 1.1

[109] Nikola T. Markov, Julien Vezoli, Pascal Chameau, Arnaud Falchier, René Quilodran, Cyril Huissoud, Camille Lamy, Pierre Misery, Pascale Giroud, Shimon Ullman, Pascal Barone, Colette Dehay, Kenneth Knoblauch, and Henry Kennedy. Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex. *Journal of Comparative Neurology*, 522(1):225–259, 2014. doi: https://doi.org/10.1002/cne.23458. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cne.23458`. 4.1

[110] D Marr and T Poggio. Cooperative Computation of Stereo Disparity. *Science*, 194(4262): pp. 283–287, 1976. (document), 1.2.1, 2.1, 2.2.2.1, 2.3.2, 2.3.2, 2.7, 2.4.1

[111] P. McCullagh and J.A. Nelder. *Generalized Linear Models, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1989. ISBN 9780412317606. URL `https://books.google.com/books?id=h9kFH2_FfBkC`. 3.3.3

[112] James M McFarland, Yuwei Cui, and Daniel A Butts. Inferring Nonlinear Neuronal Computation Based on Physiologically Plausible Inputs. *PLoS computational biology*, 9(7): e1003143, July 2013. doi: 10.1371/journal.pcbi.1003143. URL `http://dx.plos.org/10.1371/journal.pcbi.1003143`. 3.1, 3.5.2.0.1, 3.6.1, 4.5.2.2

[113] Lane T McIntosh, Niru Maheswaranathan, Aran Nayebi, Surya Ganguli, and Stephen A Baccus. Deep Learning Models of the Retinal Response to Natural Scenes. *ArXiv e-prints*, q-bio.NC, February 2017. URL `https://arxiv.org/abs/1702.01825`. 3.1, 3.2.1.0.1, 3.3.5, 3.6.3, A.1.3, A.1.5

[114] Michael D Menz and Ralph D Freeman. Stereoscopic depth processing in the visual cortex: a coarse-to-fine mechanism. *Nature Neuroscience*, 6(1):59–65, January 2003. 2.1

[115] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, 2015. doi: 10.1109/CVPR.2015.7298958. 4.5.2.1

[116] Aran Nayebi, Daniel Bear, Jonas Kubilius, Kohitij Kar, Surya Ganguli, David Sussillo, James J. DiCarlo, and Daniel L. K. Yamins. Task-driven convolutional recurrent models of the visual system. *CoRR*, abs/1807.00053, 2018. URL `http://arxiv.org/abs/1807.00053`. 1.1, 4.2.1, 4.5.1, 4.5.2.1, 4.5.3.4

[117] Emre Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, 7, 2014. 2.4.2

[118] Shani Offen, Denis Schluppeck, and David J. Heeger. The role of early visual cortex in visual short-term memory and visual attention. *Vision Research*, 49(10): 1352–1362, 2009. ISSN 0042-6989. doi: https://doi.org/10.1016/j.visres.2007.12.

022. URL https://www.sciencedirect.com/science/article/pii/S0042698908000151. Visual Attention: Psychophysics, electrophysiology and neuroimaging. 1.1

[119] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization. (document), 3.1, 3.5.1.0.2, 3.5.4, 3.7, 3.11

[120] Timothy D Oleskiw and Eero P Simoncelli. A canonical computational model of cortical area V2. *Journal of Vision*, 19(10):14b–14b, September 2019. URL https://doi.org/10.1167/19.10.14b. 1.1

[121] Randall O'Reilly, Dean Wyatte, Seth Herd, Brian Mingus, and David Jilk. Recurrent processing during object recognition. *Frontiers in Psychology*, 4:124, 2013. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00124. URL https://www.frontiersin.org/article/10.3389/fpsyg.2013.00124. 1.1

[122] Liam Paninski. Convergence properties of three spike-triggered analysis techniques. *Network: Computation in Neural Systems*, 14(3):437–464, January 2003. doi: 10.1088/0954-898X_14_3_304. URL http://www.tandfonline.com/doi/full/10.1088/0954-898X_14_3_304. A.1.5

[123] Liam Paninski. Maximum likelihood estimation of cascade point-process neural encoding models. *Network: Computation in Neural Systems*, 15(4):243–262, January 2004. doi: 10.1088/0954-898X_15_4_002. URL http://www.tandfonline.com/doi/full/10.1088/0954-898X_15_4_002. 3.3.3

[124] Il Memming Park and Jonathan W Pillow. Bayesian Spike-Triggered Covariance Analysis. In John Shawe Taylor, Richard S Zemel, Peter L Bartlett, Fernando C N Pereira, and Kilian Q Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 1692–1700, 2011. URL http://papers.nips.cc/paper/4411-bayesian-spike-triggered-covariance-analysis. 3.3.3.3, 3.3.3.3, 3.6.1

[125] Il Memming Park, Evan Archer, Nicholas Priebe, and Jonathan W Pillow. Spectral methods for neural characterization using generalized quadratic models. In Christopher J C Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2454–2462, 2013. URL http://papers.nips.cc/paper/4993-spectral-methods-for-neural-characterization-using-generalized-qu 3.1, 3.3.3.3, 3.3.3.3

[126] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani,

Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv e-prints*, art. arXiv:1912.01703, December 2019. 3.4.1.2, 3.4.2, 4.3.3.3

[127] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011. B.3.2.2

[128] Jonathan W Pillow, Jonathon Shlens, Liam Paninski, Alexander Sher, Alan M Litke, E J Chichilnisky, and Eero P Simoncelli. Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature*, 454(7207):995–999, July 2008. doi: 10.1038/nature07140. URL `http://www.nature.com/doifinder/10.1038/nature07140`. 3.1, 4.5.2.2

[129] G F Poggio and B Fischer. Binocular interaction and depth sensitivity in striate and prestriate cortex of behaving rhesus monkey. *Journal of Neurophysiology*, 40(6):1392–1405, November 1977. (document), 2.2

[130] G F Poggio, F Gonzalez, and F Krause. Stereoscopic mechanisms in monkey visual cortex: binocular correlation and disparity selectivity. *The Journal of Neuroscience*, 8 (12):4531–4550, December 1988. (document), 2.2

[131] K Prazdny. Detection of binocular disparities. *Biological Cybernetics*, 52(2):93–99, June 1985. 2.3.2

[132] Ryan Prenger, Michael C K Wu, Stephen V David, and Jack L Gallant. Nonlinear V1 responses to natural scenes revealed by neural network analysis. *Neural Networks*, 17(5-6):663–679, June 2004. doi: 10.1016/j.neunet.2004.03.008. URL `http://linkinghub.elsevier.com/retrieve/pii/S0893608004000826`. 3.1, 4.5.2.2

[133] Karim Rajaei, Yalda Mohsenzadeh, Reza Ebrahimpour, and Seyed-Mahdi Khaligh-Razavi. Beyond core object recognition: Recurrent processes account for object recognition under occlusion. *PLOS Computational Biology*, 15(5):1–30, 05 2019. doi: 10.1371/journal.pcbi.1007001. URL `https://doi.org/10.1371/journal.pcbi.1007001`. 1.1, 4.5.3.3

[134] Rajesh P N Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, January 1999. doi: 10.1038/4580. URL `http://www.nature.com/articles/nn0199_79`. 1.1, 4.2.1

[135] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 11 1999. URL `http://dx.doi.org/10.1038/14819`. 3.6.1

[136] Dario L. Ringach. Mapping receptive fields in primary visual cortex. *The Journal of Physiology*, 558(3):717–728, 2004. doi: https://doi.org/10.1113/jphysiol.2004.065771. URL `https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.2004.065771`. 4.3.1.1

[137] Dario L Ringach, Michael J Hawken, and Robert Shapley. Dynamics of orientation tuning in macaque primary visual cortex. *Nature*, 387(6630):281–284, 1997. 4.5.2.1, 4.5.2.2, 4.5.3.3

[138] Harold Rockwell, Tai Sing Lee, Yimeng Zhang, Gaya Mohankumar, and Stephen Tsou. Recurrent networks fitting neural temporal responses to natural images exhibit contextual modulation. In *Computational and Systems Neuroscience (Cosyne) Abstracts 2021*, 2021. 4.5.2.2, 4.5.3.3

[139] Ryan J Rowekamp and Tatyana O Sharpee. Cross-orientation suppression in visual area V2. *Nature Communications*, 8:15739, June 2017. doi: 10.1038/ncomms15739. URL `http://www.nature.com/doifinder/10.1038/ncomms15739`. 3.1

[140] Christopher J. Rozell, Don H. Johnson, Richard G. Baraniuk, and Bruno A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20(10):2526–2563, 2008. doi: 10.1162/neco.2008.03-07-486. URL `https://doi.org/10.1162/neco.2008.03-07-486`. 4.5.2.2

[141] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. 4.3.2.1, B.1.2.2.1

[142] Nicole C Rust and J Anthony Movshon. In praise of artifice. *Nature Neuroscience*, 8 (12):1647–1650, December 2005. doi: 10.1038/nn1606. URL `http://www.nature.com/articles/nn1606`. 1.1

[143] Nicole C Rust, Odelia Schwartz, J Anthony Movshon, and Eero P Simoncelli. Spatiotemporal Elements of Macaque V1 Receptive Fields. *Neuron*, 46(6):945–956, June 2005. doi: 10.1016/j.neuron.2005.05.021. URL `http://linkinghub.elsevier.com/retrieve/pii/S089662730500468X`. 3.1, 3.2.1.0.1, 3.3.2.1, 3.3, 3.3.2.2, 3.3.2.3, 3.6.1, A.1.5

[144] Inés Samengo and Tim Gollisch. Spike-triggered covariance: geometric proof, symmetry properties, and extension beyond Gaussian stimuli. *Journal of Computational Neuroscience*, 34(1):137–161, 2013. doi: 10.1007/s10827-012-0411-y. URL `http://link.springer.com/10.1007/s10827-012-0411-y`. A.1.5

[145] Jason M Samonds, Brian R Potetz, and Tai Sing Lee. Cooperative and Competitive Interactions Facilitate Stereo Computations in Macaque Primary Visual Cortex. *The Journal of Neuroscience*, 29(50):15780–15795, December 2009. (document), 1.2.1, 2.1, 2.2.2.1, 2.2.3.1, 2.2.3.2, 2.2.3.2, 2.2.3.2, 2.3.3, 2.8, 2.4.1

[146] Jason M Samonds, Brian R Potetz, and Tai Sing Lee. Relative luminance and binocular disparity preferences are correlated in macaque primary visual cortex, matching natural scene statistics. *Proceedings of the National Academy of Sciences*, 109(16):6313–6318, April 2012. 2.2.3.1

[147] Jason M Samonds, Brian R Potetz, Christopher W Tyler, and Tai Sing Lee. Recurrent Connectivity Can Account for the Dynamics of Disparity Processing in V1. *The Journal*

of Neuroscience, 33(7):2934–2946, February 2013. 1.2.1, 2.1, 2.2.2.1, 2.2.3.1, 2.3.2, 2.3.2, 2.4.1, 2.4.2

[148] Jason M Samonds, Christopher W Tyler, and Tai Sing Lee. Evidence of stereoscopic surface disambiguation and interpolation in the responses of V1 neurons. submitted, 2015. 2.4.1, 2.4.3

[149] Oliver Schoppe, Nicol S. Harper, Ben D. B. Willmore, Andrew J. King, and Jan W. H. Schnupp. Measuring the performance of neural models. *Frontiers in Computational Neuroscience*, 10:10, 2016. ISSN 1662-5188. doi: 10.3389/fncom.2016.00010. URL `https://www.frontiersin.org/article/10.3389/fncom.2016.00010`. 4.3.3.4, 4.3.3.4, B.4

[150] Oliver Schoppe, Nicol S Harper, Ben D B Willmore, Andrew J King, and Jan W H Schnupp. Measuring the Performance of Neural Models. *Frontiers in Computational Neuroscience*, 10:1929, February 2016. doi: 10.3389/fncom.2016.00010. URL `http://journal.frontiersin.org/Article/10.3389/fncom.2016.00010/abstract`. 3.3.6, 3.3.6

[151] M Sigman, G A Cecchi, C D Gilbert, and M O Magnasco. On a common circle: Natural scenes and Gestalt rules. *Proceedings of the National Academy of Sciences*, 98(4):1935–1940, February 2001. 2.4.1

[152] K Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, cs.CV, September 2014. URL `http://arxiv.org/abs/1409.1556`. 3.1, 3.3.5

[153] M A Smith and A Kohn. Spatial and Temporal Scales of Neuronal Correlation in Primary Visual Cortex. *The Journal of Neuroscience*, 28(48):12591–12603, November 2008. 2.3, 2.3.3

[154] Courtney J. Spoerer, Patrick McClure, and Nikolaus Kriegeskorte. Recurrent convolutional neural networks: A better model of biological object recognition. *Frontiers in Psychology*, 8:1551, 2017. ISSN 1664-1078. doi: 10.3389/fpsyg.2017.01551. URL `https://www.frontiersin.org/article/10.3389/fpsyg.2017.01551`. 1.1, 4.5.3.3

[155] Courtney J. Spoerer, Tim C. Kietzmann, Johannes Mehrer, Ian Charest, and Nikolaus Kriegeskorte. Recurrent neural networks can explain flexible trading of speed and accuracy in biological vision. *PLOS Computational Biology*, 16(10):1–27, 10 2020. doi: 10.1371/journal.pcbi.1008215. URL `https://doi.org/10.1371/journal.pcbi.1008215`. 1.2.3, 4.1, 4.2.1, 4.3.1.2, 4.3.1.2.1, 4.3.3.3.1, 4.5.2.1, 4.5.3.1

[156] M W Spratling. Predictive Coding as a Model of Response Properties in Cortical Area V1. *Journal of Neuroscience*, 30(9):3531–3543, March 2010. doi: 10.1523/JNEUROSCI.4911-09.2010. URL `http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.4911-09.2010`. 1.1, 4.2.1

[157] M W Spratling. A single functional model accounts for the distinct properties of suppression in cortical area V1. *Vision Research*, 51(6):563–576, March 2011. doi: 10.1016/j.visres.2011.01.017. URL `http://linkinghub.elsevier.com/retrieve/`

pii/S004269891100040X. 1.1, 4.2.1

[158] M W Spratling, K De Meyer, and R Kompass. Unsupervised Learning of Overlapping Image Components Using Divisive Input Modulation. *Computational Intelligence and Neuroscience*, 2009:1–19, 2009. doi: 10.1155/2009/381457. URL http://www.hindawi.com/journals/cin/2009/381457/. 1.1, 4.2.1

[159] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387. 4.2.2

[160] Shiming Tang, Tai Sing Lee, Ming Li, Yimeng Zhang, Yue Xu, Fang Liu, Benjamin Teo, and Hongfei Jiang. Complex Pattern Selectivity in Macaque Primary Visual Cortex Revealed by Large-Scale Two-Photon Imaging. *Current Biology*, 28(1):38–48.e3, January 2018. doi: 10.1016/j.cub.2017.11.039. URL http://linkinghub.elsevier.com/retrieve/pii/S096098221731521X. (document), 1.1, 1.2.2, 3.1, 3.2.1, 3.2.1.0.1, 3.2.2.0.1, 3.2.2.0.2, 3.1, 3.5.3.0.2, 3.5.4, 3.6.1, 3.6.2, A.1.5

[161] Shiming Tang, Yimeng Zhang, Zhihao Li, Ming Li, Fang Liu, Hongfei Jiang, and Tai Sing Lee. Large-scale two-photon imaging revealed super-sparse population codes in the v1 superficial layer of awake monkeys. *eLife*, 7:e33370, apr 2018. ISSN 2050-084X. doi: 10.7554/eLife.33370. URL https://doi.org/10.7554/eLife.33370. 4.3.2.2, B.1.2.2.2, B.3.2.2

[162] M F Tappen and W T Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, pages 900–906, 2003. 2.2.2.1

[163] F E Theunissen, Stephen V David, N C Singh, A Hsu, W E Vinje, and Jack L Gallant. Estimating spatio-temporal receptive fields of auditory and visual neurons from their responses to natural stimuli. *Network: Computation in Neural Systems*, 12(3):289–316, January 2001. doi: 10.1080/net.12.3.289.316. URL http://www.tandfonline.com/doi/full/10.1080/net.12.3.289.316. 3.1

[164] Jon Touryan, Gidon Felsen, and Yang Dan. Spatial Structure of Complex Cell Receptive Fields Measured with Natural Images. *Neuron*, 45(5):781–791, March 2005. doi: 10.1016/j.neuron.2005.01.029. URL http://linkinghub.elsevier.com/retrieve/pii/S0896627305000619. 3.1

[165] D. C. Van Essen, W. T. Newsome, J. H. R. Maunsell, and J. L. Bixby. The projections from striate cortex (v1) to areas v2 and v3 in the macaque monkey: Asymmetries, areal boundaries, and patchy connections. *Journal of Comparative Neurology*, 244(4):451–480, 1986. doi: 10.1002/cne.902440405. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cne.902440405. 1.1

[166] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–

6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html. 4.5.2.5

[167] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 550–558. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/37bc2f75bf1bcfe8450a1a41c200364c-Paper.pdf. 4.2.2, 4.3.4

[168] Valérie Ventura, Can Cai, and Robert E Kass. Statistical Assessment of Time-Varying Dependency Between Two Neurons. *Journal of Neurophysiology*, 94(4):2940–2947, April 2005. 2.2.3.2

[169] Valérie Ventura, Can Cai, and Robert E Kass. Trial-to-Trial Variability and Its Effect on Time-Varying Dependency Between Two Neurons. *Journal of Neurophysiology*, 94(4): 2928–2939, April 2005. 2.2.3.2

[170] Petra Vetter, Łukasz Bola, Lior Reich, Matthew Bennett, Lars Muckli, and Amir Amedi. Decoding natural sounds in early "visual" cortex of congenitally blind individuals. *Current Biology*, 30(15):3039–3044.e2, 2020. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2020.05.071. URL https://www.sciencedirect.com/science/article/pii/S0960982220307582. 1.1

[171] Jonathan D. Victor, Ferenc Mechler, Michael A. Repucci, Keith P. Purpura, and Tatyana Sharpee. Responses of v1 neurons to two-dimensional hermite functions. *Journal of Neurophysiology*, 95(1):379–400, 2006. ISSN 0022-3077. doi: 10.1152/jn.00498.2005. URL http://jn.physiology.org/content/95/1/379. 1.1, 3.1

[172] Brett Vintch, J. Anthony Movshon, and Eero P. Simoncelli. A convolutional subunit model for neuronal responses in macaque v1. *Journal of Neuroscience*, 35(44):14829–14841, 2015. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.2815-13.2015. URL http://www.jneurosci.org/content/35/44/14829. 3.1, 3.5.2.0.1, 3.6.1, 4.5.2.2, A.1.5

[173] H von Helmholtz. *Handbuch der physiologischen Optik*. Leipzig Voss, 1896. 2.1

[174] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7794–7803. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00813. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Wang_Non-Local_Neural_Networks_CVPR_2018_paper.html. 4.5.2.5

[175] Max Welling and Geoffrey E Hinton. A New Learning Algorithm for Mean Field Boltzmann Machines. In *International Conference on Artificial Neural Networks 2002*, pages 351–357, 2002. 2.2.2.1

[176] Ben D. B. Willmore, Ryan J. Prenger, and Jack L. Gallant. Neural representation of natural images in visual area v2. *Journal of Neuroscience*, 30(6):2102–2114, 2010. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.4099-09.2010. URL https://www.jneurosci.org/content/30/6/2102. 1.1, 4.5.2.2

[177] Michael C K Wu, Stephen V David, and Jack L Gallant. Complete Functional Characterization of Sensory Neurons by System Identification. *Annual Review of Neuroscience*, 29(1):477–505, July 2006. doi: 10.1146/annurev.neuro.29.051605.113024. URL `http://www.annualreviews.org/doi/10.1146/annurev.neuro.29.051605.113024`. 3.1, 3.6.1

[178] Daniel Yamins, Ha Hong, Charles F Cadieu, and James J DiCarlo. Hierarchical Modular Optimization of Convolutional Networks Achieves Representations Similar to Macaque IT and Human Ventral Stream. In Christopher J C Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3093–3101, 2013. URL `http://papers.nips.cc/paper/4991-hierarchical-modular-optimization-of-convolutional-networks-achie` 3.1, 3.6.2

[179] Daniel L K Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, 19(3):356–365, February 2016. doi: 10.1038/nn.4244. URL `http://www.nature.com/doifinder/10.1038/nn.4244`. 1.1, 1.2.3, 3.1, 3.3.5, 3.6.2, 4.1, 4.2.1

[180] Christoph Zetzsche and Florian Röhrbein. Nonlinear and extra-classical receptive field properties and the statistics of natural scenes. *Network: Computation in Neural Systems*, 12(3):331–350, 2001. doi: 10.1080/net.12.3.331.350. URL `http://www.tandfonline.com/doi/full/10.1080/net.12.3.331.350`. 1.1, 4.2.1

[181] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1822–1830, 2016. URL `https://proceedings.neurips.cc/paper/2016/hash/860320be12a1c050cd7731794e231bd3-Abstract.html`. 4.2.2, 4.5.2.3

[182] Yimeng Zhang, Xiong Li, Jason M. Samonds, and Tai Sing Lee. Relating functional connectivity in v1 neural circuits and 3d natural scenes using boltzmann machines. *Vision Research*, 120:121 – 131, 2016. ISSN 0042-6989. doi: https://doi.org/10.1016/j.visres.2015.12.002. URL `http://www.sciencedirect.com/science/article/pii/S004269891500365X`. Vision and the Statistics of the Natural Environment. 1.2.1

[183] Yimeng Zhang, Corentin Massot, Tiancheng Zhi, George Papandreou, Alan Yuille, and Tai Sing Lee. Understanding neural representations in early visual areas using convolutional neural networks. In *Neuroscience (SfN)*, 2016. 1

[184] Yimeng Zhang, Tai Sing Lee, Ming Li, Fang Liu, and Shiming Tang. Convolutional neural network models of v1 responses to complex patterns. *Journal of Computational Neuroscience*, 46(1):33–54, Feb 2019. ISSN 1573-6873. doi: 10.1007/s10827-018-0687-7. URL `https://doi.org/10.1007/s10827-018-0687-7`.

1.1, 1.2.2, 4.1, 4.3.3.4, 4.5.2.2

[185] Mengchen Zhu and Christopher J Rozell. Visual Nonclassical Receptive Field Effects Emerge from Sparse Coding in a Dynamical System. *PLoS computational biology*, 9(8): e1003191, August 2013. doi: 10.1371/journal.pcbi.1003191. URL `http://dx.plos.org/10.1371/journal.pcbi.1003191`. 1.1, 4.2.1, 4.5.2.2, 4.5.3.3

[186] Steven W Zucker. Stereo, Shading, and Surfaces: Curvature Constraints Couple Neural Computations. *Proceedings of the IEEE*, 102(5):812–829, 2014. 2.4.2