Journal of Computational Neuroscience manuscript No.
(will be inserted by the editor)

Supplementary Materials for *Convolutional neural network models of V1 responses to complex patterns*

Yimeng Zhang* · **Tai Sing Lee** · **Ming Li** · **Fang Liu** · **Shiming Tang***

A Results

A.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.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 3.4 and Table 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 Cadena et al. (2017); Klindt et al. (2017). 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

* YZ and ST are co-corresponding authors.
YZ and TL
Center for the Neural Basis of Cognition and Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213

ML, FL, and ST
Peking University School of Life Sciences and Peking-Tsinghua Center for Life Sciences, Beijing 100871, China
IDG/McGovern Institute for Brain Research at Peking University, Beijing 100871, China

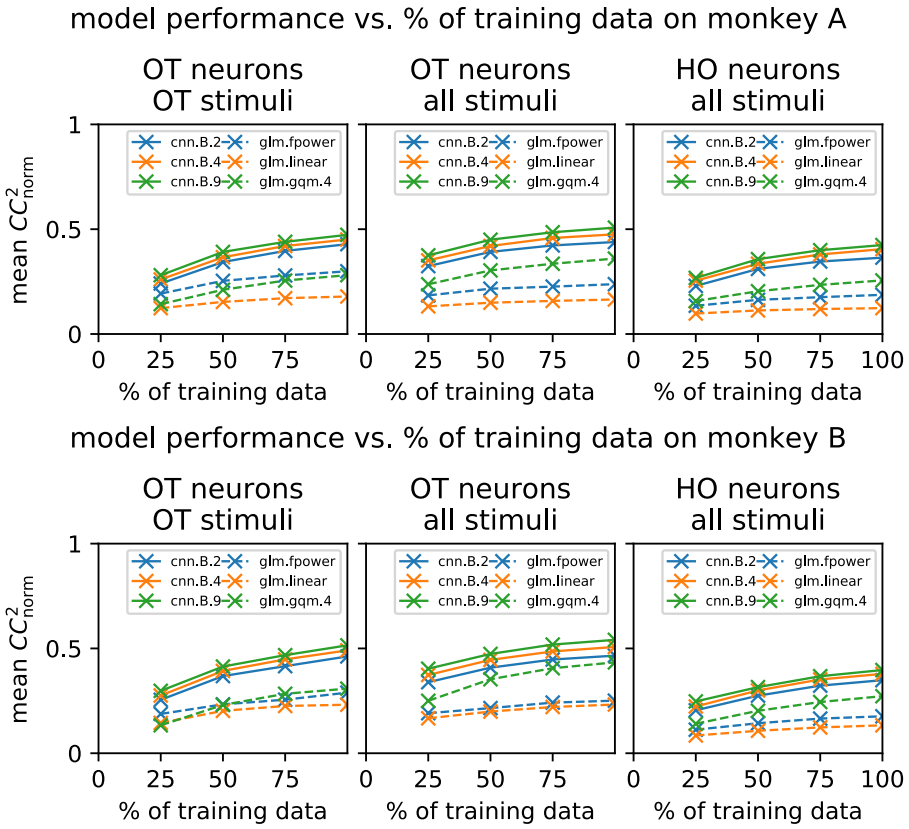

Fig. A.1 CNN models vs. others with different amounts of training data. The organization of panels is the same as that in Figure 6. For each panel, the model performance is shown in CC^2_{norm} averaged over neurons in the neuron subset, as a function of the amount of data used for training and validation. 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.

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 3.6); in particular, from each of the seven neuron subclasses (Section 2.2), 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 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 pool-

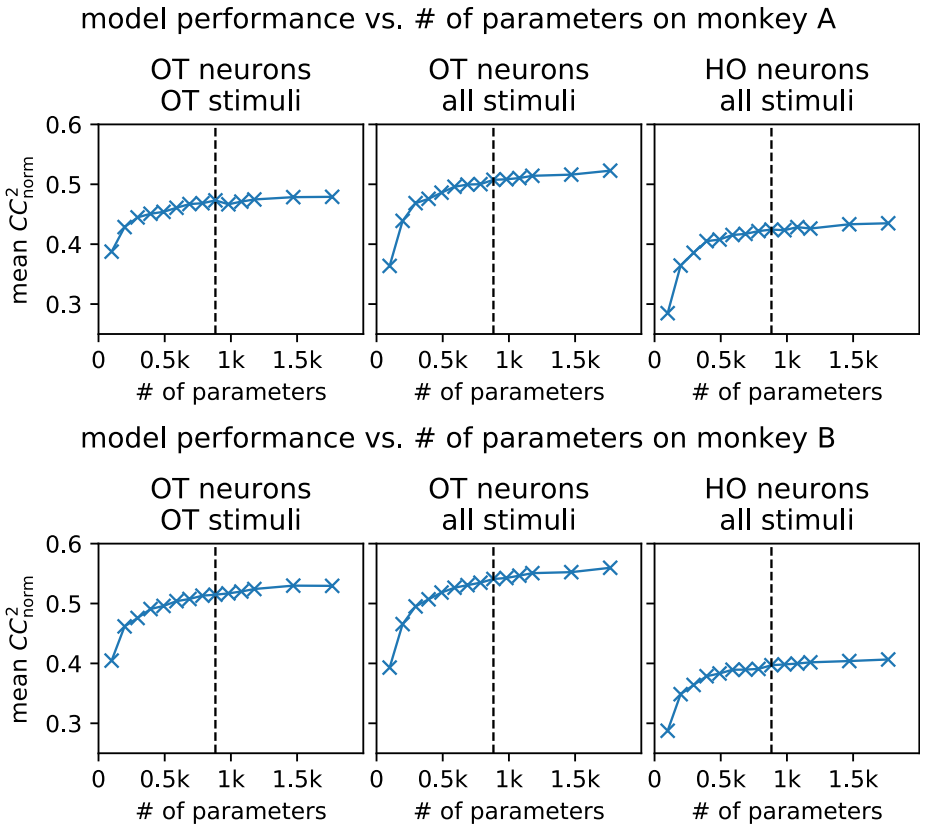

Fig. A.2 Performance vs. number of parameters for CNN models. The organization of panels is the same as that in Figure 6. For each panel, the model performance is shown in CC^2_{norm} averaged over neurons in the neuron subset, as a function of the number of parameters in the model; the vertical line denotes the number of parameters for the baseline CNN model in the main text.

ing layer of configuration `k6s2` worked better than other schemes (other pooling configurations or factorized readout layer). Accordingly, we chose the baseline CNN architecture (Figure 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).

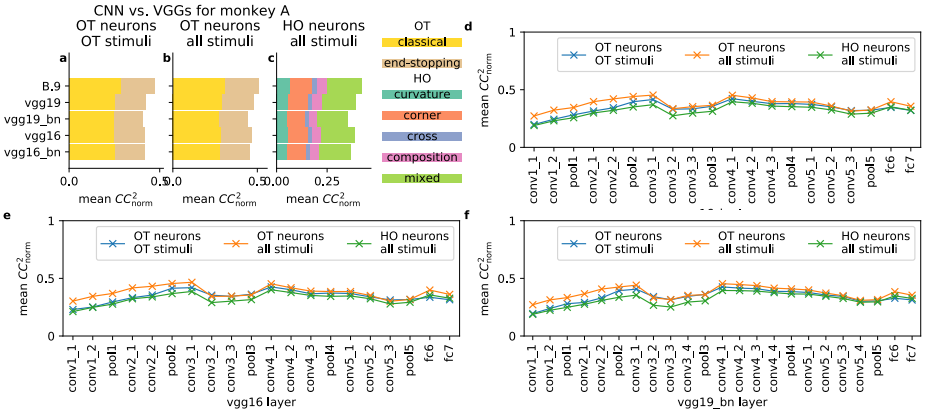

Fig. A.3 Transfer learning (goal-driven) approach for modeling V1 neurons using pre-trained VGG networks, monkey A. These panels have similar formats to those in Figure 11. **a-c** VGG networks' best performing layers (`conv3_1` for all) vs. the baseline CNN (`B.9`). **d-f** Model performance across layers for different VGG networks (**d** for VGG16 with batch normalization, **e** for VGG16, **f** for VGG19 with batch normalization).

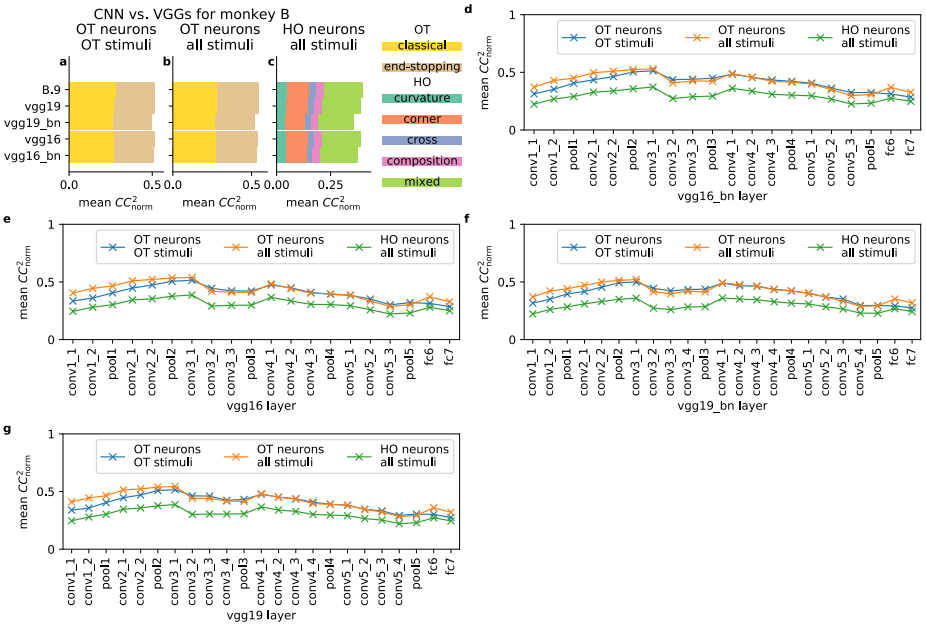

Fig. A.4 Transfer learning (goal-driven) approach for modeling V1 neurons using pre-trained VGG networks, monkey B. Compared to Figure A.3, additional VGG19 results (**g**) are shown.

A.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 (Kindel et al., 2017; McIntosh et al., 2017; Klindt et al., 2017; Cadena et al., 2017). As a preliminary attempt in this direction, we tried modeling monkey A's neurons using a single

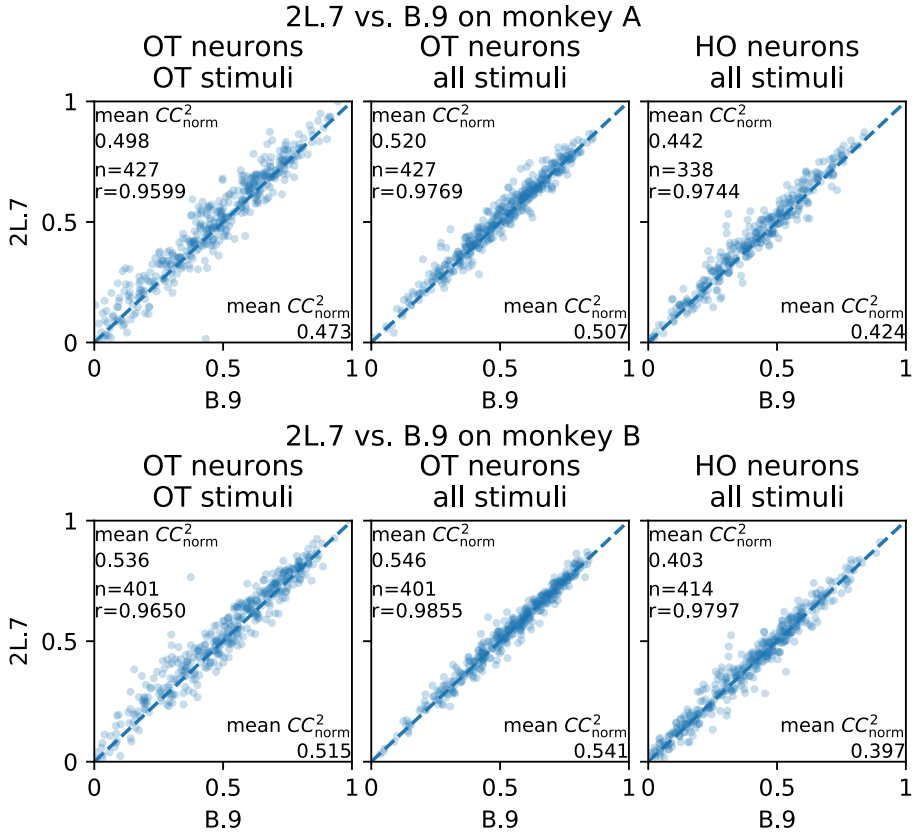

Fig. A.5 Neuron-by-neuron comparison of our best candidate two-convolutional-layer CNN (`2L.7`) vs. the baseline CNN (`B.9`). The organization of panels is the same as that in Figure 6. For each panel, we show the scatter plot of the two models' performance on individual neurons in terms of CC^2_{norm} (Section 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).

CNN. The architecture of our CNN is the the same as that in Klindt et al. (2017); Cadena et al. (2017), 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.

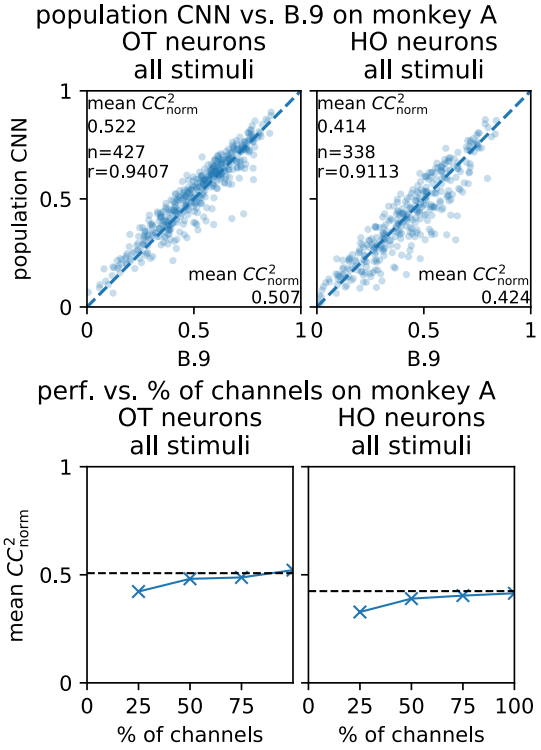

Fig. A.6 Performance of modeling multiple neurons using a single CNN, Monkey A. The comparison of our single CNN (`population CNN`) and our baseline CNN (`B.9`) are shown in the top. Only cases for OT neurons, all stimuli and HO neurons, all stimuli were explored due to limited time. For either HO or 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.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.

A.4 Other output nonlinearities of GLMs

In the main text (Section 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 (Goodfellow et al., 2016) 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.

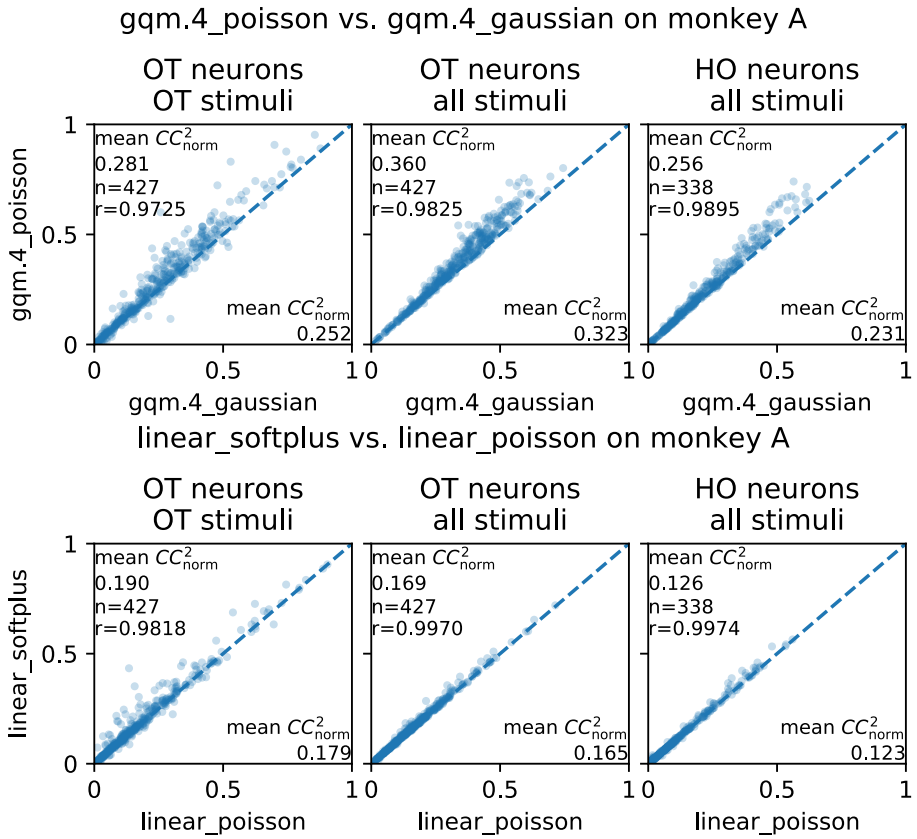

Fig. A.7 Neuron-by-neuron comparison of Gaussian, Poisson, and softplus GLMs. The Gaussian and Poisson versions of the GQM with locality 4 (`gqm.4`) were shown in the top, and the Poisson and softplus versions of the vanilla GLM were shown in the bottom. The organization of panels is the same as that in Figure 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 CC^2_{norm} (Section 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.

A.5 Other possible stimulus types

Much previous work modeling V1 neurons used natural images or natural movies (Kindel et al., 2017; Cadena et al., 2017; David and Gallant, 2005), while we used artificial pattern images (Tang et al., 2018). 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 (Coen-Cagli et al., 2015) 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 2.2). While white noise stimuli (Rust et al., 2005; McIntosh et al., 2017) 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 Paninski (2003); Samengo and Gollisch (2013) 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. (2015).

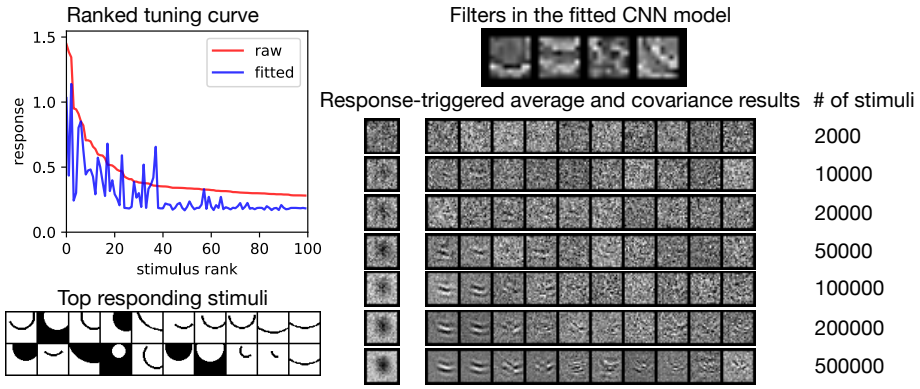

Fig. A.8 Failure to recover CNN model parameters using a reasonable amount of white noise stimuli. Left: neuron 554 of monkey A was first fitted by a CNN model with one convolutional layer of four filters (architecture `B.4` in Figure 6). Right: standard response-triggered average and covariance methods 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.

A.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 7, in a similar format.

Comparing this figure with Figure 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 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.

References

Cadena, S. A., Denfield, G. H., Walker, E. Y., Gatys, L. A., Tolias, A. S., Bethge, M., and Ecker, A. S. (2017). Deep convolutional models improve predictions of macaque v1 responses to natural images. *bioRxiv*.

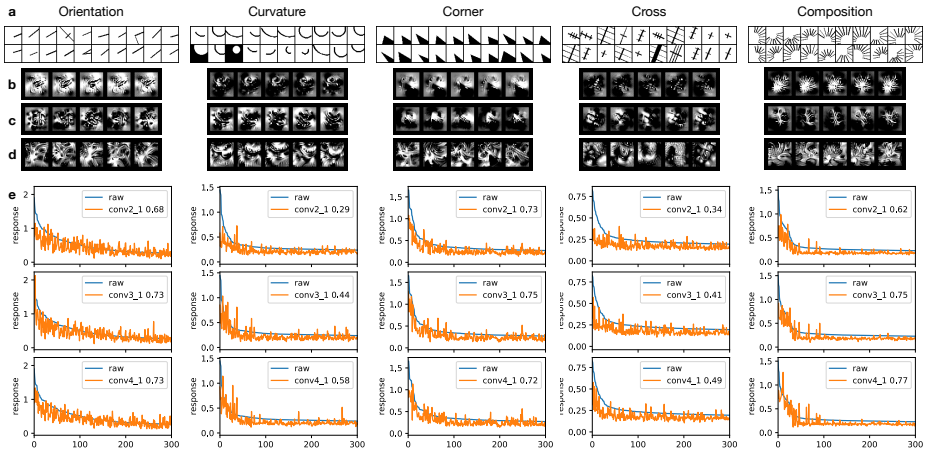

Fig. A.9 Example neurons and their fitting results using VGG19. The example neurons are the same as those in Figure 7 and two figures have similar formats. For each of the five columns, we show the following information (**a-e**). **a** The top 20 responding stimuli of the neuron; **b-d** visualization 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`.

Coen-Cagli, R., Kohn, A., and Schwartz, O. (2015). Flexible gating of contextual influences in natural vision. *Nature Neuroscience*, 18:1648 EP –.

David, S. V. and Gallant, J. L. (2005). Predicting neuronal responses during natural vision. *Network: Computation in Neural Systems*, 16(2-3):239–260.

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

Kindel, W. F., Christensen, E. D., and Zylberberg, J. (2017). Using deep learning to reveal the neural code for images in primary visual cortex. *ArXiv e-prints*, q-bio.NC.

Klindt, D., Ecker, A. S., Euler, T., and Bethge, M. (2017). Neural system identification for large populations separating "what" and "where". In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., 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.

McIntosh, L. T., Maheswaranathan, N., Nayebi, A., Ganguli, S., and Baccus, S. A. (2017). Deep Learning Models of the Retinal Response to Natural Scenes. *ArXiv e-prints*, q-bio.NC.

Paninski, L. (2003). Convergence properties of three spike-triggered analysis techniques. *Network: Computation in Neural Systems*, 14(3):437–464.

Rust, N. C., Schwartz, O., Movshon, J. A., and Simoncelli, E. P. (2005). Spatiotemporal Elements of Macaque V1 Receptive Fields. *Neuron*, 46(6):945–956.

Samengo, I. and Gollisch, T. (2013). Spike-triggered covariance: geometric proof, symmetry properties, and extension beyond Gaussian stimuli. *Journal of Computational Neuroscience*, 34(1):137–161.

Tang, S., Lee, T. S., Li, M., Zhang, Y., Xu, Y., Liu, F., Teo, B., and Jiang, H. (2018). Complex Pattern Selectivity in Macaque Primary Visual Cortex Revealed by Large-Scale Two-Photon Imaging. *Current Biology*, 28(1):38–48.e3.

Vintch, B., Movshon, J. A., and Simoncelli, E. P. (2015). A convolutional subunit model for neuronal responses in macaque v1. *Journal of Neuroscience*, 35(44):14829–14841.