An invariance-based account of feedforward categorization in a realistic model of the ventral visual pathway

by

Jim Mutch

Submitted to the Department of Brain and Cognitive Sciences in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY September 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Signature redacted

Author

Department of Brain and Cognitive Sciences September 2, 2016

Signature redacted

Certified by

Tomaso A. Poggio Eugene McDermott Professor, Brain and Cognitive Sciences Thesis Supervisor

Signature redacted

Accepted by

Matthew A. Wilson Sherman Fairchild Professor of Neuroscience and Picower Scholar Director of Graduate Education for Brain and Cognitive Sciences
An invariance-based account of feedforward categorization in a realistic model of the ventral visual pathway

by

Jim Mutch

Submitted to the Department of Brain and Cognitive Sciences on September 2, 2016, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

For the recognition of general objects in natural scenes, the current top-performing computer vision models owe a debt to visual neuroscience. The hierarchical architecture of convolutional networks, and related models such as HMAX, mimics that of the ventral stream of visual cortex. In essence, they apply the model of Hubel and Wiesel recursively, alternating layers of 'simple' cells, which are tuned to certain local features, and 'complex' cells, which pool the outputs of simple cells within a local region.

With recent advances in deep learning, for many tasks in vision and speech, emphasis has moved away from so-called 'hand-designed' models and toward big data and high throughput computing, with models learning from millions of labeled examples. Yet CNNs only learn their features — the weights of connections in the network. All other aspects of the network (size, connectivity, response functions, etc.) are unlearned architectural choices made by their designers. Vision has not yet been reduced to a pure learning problem — human insight into the nature of visual problems continues to be important. To design a good vision system, one still has to understand vision. And, as evidenced by performance for many complex visual tasks, natural vision systems still 'understand' vision better than we do; there is still much to be learned from them.

Our work is based on the HMAX model, which places greater weight on biological realism. Our goals are threefold: to better understand the ventral stream algorithm, as well as the visual problem it solves, and to improve the performance of artificial vision systems. In this work we take two main approaches.

i-theory is an ongoing effort to explain the good performance of hierarchical models in terms of a formal theory of invariance to transformations. We provide a reinterpretation of V1 simple and complex cells in the context of i-theory as computing a high-dimensional, locally translation-invariant signature for the contents of a V1 receptive field. We describe a simple algorithm for learning them which can extend without modification to the learning of higher-order representations for V2 and beyond. The algorithm yields model V1 cells having a good fit to data from several
animal species. We also demonstrate that a precondition of i-theory, covariance, can hold in upper layers, even for transformations not anticipated in the training of lower layers.

No current hierarchical object recognition model incorporates realistic retinal resolution. Incorporating this detail forces a reevaluation of the role of the ventral stream's feedforward core in the larger task of scene understanding as well as many details of the model itself, particularly with respect to scale. We investigate the optimal shape of the input window used to select a subset of the visual information available in a scene for processing in a single feedforward pass, defined as a region in \((x, y, \lambda)\), the handling of the \(\lambda\) dimension within the hierarchy, and the problem of clutter. Our main experimental results are (1) spatial wavelengths too small for the retina to perceive across the entire object do not play a significant role in the no-clutter case, but confer robustness in the presence of clutter, and (2) preservation by the hierarchy of information about the relative scale (distance along \(\lambda\)) of feature activations is more important than current models reflect.

Thesis Supervisor: Tomaso A. Poggio
Title: Eugene McDermott Professor, Brain and Cognitive Sciences
Acknowledgments

None of this work would have been possible without the remarkable creativity, breadth of knowledge, and enthusiastic support of my advisor Tomaso Poggio. Each of the studies presented here has as its starting point a question or an insight of Tommy’s.

I thank my final committee, Shimon Ullman, David Lowe, and Joshua Tenenbaum, as well as previous committee members Sebastian Seung, James DiCarlo, and Robert Desimone, for their guidance, kindness, and sincere interest in helping me to succeed.

Thank you to Matt Wilson, Director of Graduate Education, and to all the administrative staff who look out for us. Gadi Geiger, Kathleen Sullivan, Denise Heintze, Meredith Canode, and Julianne Ormerod have been especially helpful to me. Thanks also to Alex and Helder for keeping my spirits up when I worked the graveyard shift, and to Retsina for helping me across the finish line.

To fellow CBCL members present and past – Andrea, Bernd, Carlo, Caroline, Cheston, Danny, Ethan, Gemma, Georgios, Guillermo, Hilde, Hueihan, Jake, Joel, Leyla, Lorenzo, Mattia, Max, Minjoon, Neva, Owen, Pavan, Qianli, Sharat, Stan, Steve, Thomas, Tony, Ulf, Youssef, and anyone I’ll owe an apology for omitting – thanks for friendship and for talks both intellectual and non. I’ll miss you and hope to see you soon.

To all my ‘new’ Boston friends: thanks for laughter and company and a sense of belonging. (Teasing about country, and not decade, of origin was exactly the right thing.) We have a guest room and Vancouver is lovely. Thanks to friends old and new for continuing to be friends through periods of thesis-induced neglect.

To Rachel, my partner, who managed to get a whole Harvard degree while I was ‘probably six months from finishing’ this one: congratulations! You never stop impressing me. As certainly as I know anything, I know I can always count on you.

With profound gratitude for such a gift, I dedicate this thesis to you.
# Contents

1 Background ................................................. 13
   1.1 Object recognition ...................................... 13
   1.2 The ventral visual pathway .............................. 13
       1.2.1 V1 and topographical maps ......................... 14
       1.2.2 Hierarchical organization .......................... 15
       1.2.3 Immediate recognition and feedforward operation .... 15
   1.3 HMAX ...................................................... 16
       1.3.1 From input image to invariant signature .............. 16
       1.3.2 Hierarchical architecture .......................... 17
       1.3.3 Normalized dot product ............................. 19
       1.3.4 Scale dimension ................................... 20

2 i-theory ...................................................... 23
   2.1 General invariance via projection onto
       transformed templates ..................................... 23
   2.2 Efficient computation of invariant signatures using learned energy-
       based features ............................................ 26
       2.2.1 Gabor filter responses as components of a translation-invariant
           signature .............................................. 26
       2.2.2 Learning algorithm .................................. 28
       2.2.3 Limitations ......................................... 32
       2.2.4 Other transformations ............................... 33
       2.2.5 Matching V1 tuning .................................. 36
2.3 Extending the energy-based model to higher levels .......................... 36
  2.3.1 Learning in $C_1$ space ............................................. 36
  2.3.2 Implications of higher dimensionality ............................... 38
  2.3.3 Visualizing higher-order features ................................... 39
2.4 Covariant representations .................................................... 41
2.5 Summary of contributions ..................................................... 43

3 Scale-invariant object categorization under eccentricity-dependent
  retinal resolution 45
  3.1 Theory .............................................................................. 46
    3.1.1 Object recognition .................................................... 46
      3.1.1.1 Objects and labels ............................................. 46
      3.1.1.2 Hierarchies of objects and parts ............................. 47
      3.1.1.3 Parts vs. features ............................................. 48
      3.1.1.4 General transformations and invariance ..................... 49
    3.1.2 Capture and input windows .......................................... 51
      3.1.2.1 Uniform window shape ........................................ 51
      3.1.2.2 Retinal window shape ......................................... 52
      3.1.2.3 Attentional masking ........................................... 52
    3.1.3 Problem scope ............................................................ 53
    3.1.4 Invariance to object position and scale ........................... 53
    3.1.5 Invariance to within-object transformations ..................... 55
    3.1.6 Whole-part representation .......................................... 57
      3.1.6.1 Fine bands ...................................................... 58
      3.1.6.2 Coarse bands ................................................... 58
    3.1.7 'Has-a' relationships ................................................ 59
  3.2 Experiments: isolated objects .............................................. 60
    3.2.1 Model ....................................................................... 60
    3.2.2 Procedure ................................................................. 64
    3.2.3 Results ..................................................................... 64
3.2.3.1 Effect of window shape ........................................ 65
3.2.3.2 Global vs. no scale pooling ................................. 67
3.2.3.3 Effect of window shape (no scale pooling) ............ 69
3.2.3.4 Effect of band masking .................................... 70
3.2.3.5 Effect of scale pooling range .............................. 72
3.2.3.6 Summary of optimizations .................................. 73
3.3 Experiments: clutter .............................................. 74
3.4 Summary of contributions ...................................... 74

4 Discussion .................................................................. 79
4.1 General approach .................................................. 79
  4.1.1 Feedforward core ............................................. 79
  4.1.2 Biological fidelity ........................................... 80
    4.1.2.1 What should models learn? ........................... 83
  4.1.3 Next steps ..................................................... 83
4.2 Specific issues ..................................................... 84
  4.2.1 Deeper hierarchy ............................................ 84
    4.2.1.1 No shifting .............................................. 85
    4.2.1.2 BOF degree ............................................. 85
    4.2.1.3 Multiscale receptive fields ......................... 86
  4.2.2 Small or unfixated objects ................................ 86
    4.2.2.1 Pass-through concept ................................. 87
  4.2.3 Input window vs. object location ....................... 87
  4.2.4 Top-level classifier ....................................... 88
  4.2.5 Context-assisted recognition ............................. 88
  4.2.6 Connection with minimal images ....................... 88

A CNS: a GPU-based framework for simulating cortically-organized networks ......................................................... 89
A.1 Introduction ...................................................... 90
  A.1.1 Definitions .................................................. 90
A.1.2 Motivation ................................................. 90
A.1.3 Overview of CNS .......................................... 92
A.1.4 Structure of this report ................................. 93
A.2 Example packages .......................................... 94
  A.2.1 'HH' package: Hodgkin-Huxley spiking models .......... 94
  A.2.2 'FH' package: HMAX-like feature hierarchies .......... 96
  A.2.3 'CN' package: convolutional networks for 3-D image segmentation 98
A.3 CNS concepts .............................................. 99
  A.3.1 Layers and groups of layers .......................... 100
  A.3.2 Execution order .................................... 100
  A.3.3 Cell types ....................................... 101
  A.3.4 Fields and scope .................................. 101
  A.3.5 Kernels ......................................... 104
  A.3.6 Connectivity .................................... 105
A.4 Working with CNS ......................................... 108
A.5 GPU details ............................................... 109
A.6 Limitations ............................................... 111
  A.6.1 Inherent limitations ............................... 111
  A.6.2 Current limitations ............................... 112
A.7 Summary of contributions ............................... 112
A.8 Future work ............................................. 114
List of Figures

1-1 The ventral visual pathway .............................................. 14
1-2 An example HMAX hierarchy .............................................. 16

2-1 HMAX performance using random templates ................................ 24
2-2 Invariance to translation of a $C_1$ column ................................ 29
2-3 Templates learned from horizontal translation (no temporal derivative) 31
2-4 Templates learned from horizontal translation (temporal derivative) 32
2-5 Templates learned from centered rotation .................................. 34
2-6 Templates learned from centered expansion .................................. 35
2-7 Gabor filter shapes learned by our algorithm vs. animal studies .......... 37
2-8 Higher-order templates learned from horizontal translation ............... 38
2-9 Higher-order templates learned from off-center rotation .................. 39
2-10 Rotations are more shift-like in $C_1$ space ................................ 40
2-11 Responses of two theoretical $C_2$ cells to Gallant stimuli ............... 41
2-12 Covariance of $C_1$ outputs under rotation ................................ 42
2-13 Covariance of $C_2$ outputs under rotation ................................ 42

3-1 Uniform and retinal window shapes ....................................... 50
3-2 Object centered in a retinal window ....................................... 56
3-3 Matching signatures having $\lambda$ topology ................................ 62
3-4 Object sizes used in scale-invariant recognition experiments ............ 63
3-5 Results: effect of window shape ........................................... 65
3-6 Results: global vs. no scale pooling ....................................... 67
3-7 Results: effect of window shape (no scale pooling) ......................... 69
3-8 Results: effect of band masking. ........................................ 70
3-9 Results: effect of scale pooling range. .......................... 72
3-10 Results: summary of optimizations. ............................. 73
3-11 Results: clutter experiment. ........................................ 75
4-1 Marr's levels of analysis illustrated for object recognition. .. 81
4-2 Shape of multiscale receptive fields. ............................. 86
A-1 Part of a Hodgkin-Huxley spiking neuron model. ............. 95
A-2 Voltage trace from a Hodgkin-Huxley model. .................. 96
A-3 Example HMAX model. ............................................... 97
A-4 Part of a 3-D convolutional network for segmenting 3-D images. 98
A-5 Part of a kernel from the CN package. .......................... 106
A-6 Common coordinate positions for several layers of an FH model. 107
Chapter 1

Background

1.1 Object recognition

We are interested in the visual recognition of objects in images of general scenes. We limit our attention to images without color or explicit depth information, i.e., to intensity images taken with a single camera, and to nonmoving images, except during learning. (§3.1.1 goes into considerably more detail.)

1.2 The ventral visual pathway

Object recognition in cortex is centered in the ventral visual pathway, running from primary visual cortex (V1) through areas V2, V4, and inferotemporal cortex (IT) (fig. 1-1).

Primates are able to learn invariant recognition of novel object categories from few examples. The computational role of the ventral visual pathway is commonly interpreted as the conversion of a retinal image into a representation better-suited for use in a classifier, reducing as much as possible the sample complexity – the number of images of each category needed for a given level of performance.
1.2.1 V1 and topographical maps

Among all the visual areas V1 has received the most study (although it is far from being fully understood [32]). Cells in V1 respond to very simple features (essentially oriented bars) at specific retinal positions and scales [11].

V1 is one of many cortical areas with a clear topographic mapping: its 2D layout corresponds roughly to 2D retinotopic position. The additional stimulus dimensions of scale and orientation are folded into this 2D layout so that a given bit of V1 cortex will contain cells responsive to a variety of scales and orientations at a specific retinotopic position. This packing of multiple stimulus dimensions into a 2D array is done in a manner that maintains continuity along stimulus dimensions while covering the entire stimulus space [47]. Note that V1 contains additional stimulus dimensions (colour, stereo, motion) that we ignore in this study.

Topographical maps are numerous in cortex: there are about 25 of them in the visual system alone [3]. There are also topographical maps for body location in the somatosensory system, frequency in the auditory system, and muscle groups in the motor system. Continuous topographical maps minimize the amount of wiring necessary for performing local computations. Neurons representing similar stimuli are kept close together.
1.2.2 Hierarchical organization

As we move through the ventral stream from V1 through V2, V4, and IT, we encounter cells that are responsive to increasingly complex stimuli with increasing invariance to position and scale. The first step seems to occur within V1 itself, where the outputs of a number of ‘simple’ cells responsive to the same feature (i.e., a particular orientation) are pooled by ‘complex’ cells [11]. A resultant complex cell is responsive to the same orientation as its inputs but has a larger receptive field, i.e., a greater degree of position and scale invariance. Further steps combine heterogeneous features to generate more complex features, and more pooling over position and scale occurs. At the level of IT we encounter cells with a high degree of position and scale invariance which are responsive to specific object views, as well as viewpoint-invariant units.

Despite the increasing degree of spatial invariance at higher levels, combinatorical constraints on the number of complex features that could be represented rule out complete coverage of the potential stimulus space, as seems to occur in V1.

This hierarchical arrangement of topographical maps is not unique to the visual system; similar hierarchies are present in the somatosensory, auditory, and motor systems [3].

1.2.3 Immediate recognition and feedforward operation

Most forward projections between brain areas are matched by corresponding feedback connections, and the ventral stream is known to be modulated by other areas for reasons including attention and contextual priming. Nevertheless we seems to do quite well in an ‘immediate’ recognition mode in which these effects are absent. Human subjects in rapid serial visual presentation (RSVP) experiments have been able to process images as rapidly as 8 per second [38]. EEG studies [49] show response times on the order of the latency of the ventral stream, suggesting a mainly feedforward mode of operation for this first stage of the basic categorization task.

The feedforward constraint, however, does not apply to learning.
1.3 HMAX

HMAX [39, 44, 29] is a class of hierarchical models for object recognition based on the primate ventral visual pathway.

This section outlines HMAX as described in existing literature.

1.3.1 From input image to invariant signature

The role of an HMAX hierarchy is to convert an image of an object into a high-dimensional signature vector suitable for use in a classifier. In order to reduce the number of images needed to train the classifier, the signature should be as invariant as possible to the many label-preserving transformations that complicate object recognition, while remaining distinctive for the object’s correct label.

Figure 1-2: illustrates a particular HMAX hierarchy (§1.3.2). Each horizontal row of 3D boxes denotes a 4D layer; multiple scale bands (§1.3.4) within a layer are shown as separate \((x, y, f)\) boxes in the same row, ordered from smallest \(\lambda\) on the left to largest on the right (not all bands are shown). Bands for smaller \(\lambda\) are larger in \((x, y)\) because they must be sampled at smaller \((x, y)\) intervals and thus require more units to span the input image. Pooling and subsampling over \((x, y, \lambda)\) reduces the size of these dimensions; by \(C_2\) global pooling has reduced them to size 1, leaving only the feature dimension \(f\). The vector of \(C_2\) outputs is the signature for the input image and becomes the input to a final classifier (not shown).
HMAX models only the core ‘immediate’ recognition capability of the ventral stream: a single feedforward pass from input image to signature. How this capability is integrated into the larger task of full scene understanding has been beyond its scope. In §3 we take a step towards remedying that.

The signature emerges at the top level of the hierarchy. It is intended to correspond to anterior inferotemporal cortex (AIT), the highest purely visual area along the ventral visual pathway.

The final classifier itself is not intended to mimic biology and is typically not the focus of HMAX research. As such it is often restricted to being simple and linear. This ensures that performance of the overall model remains tied to the quality of the signature representation generated by the hierarchy.

Immediate recognition is, of course, not the entire story for biological vision. It is simply the mode for which we have the most physiological data, as well as the most theoretically tractable. To the extent it is seen as a ‘core capability’ which is modulated by feedback to perform more complex tasks, these may be (more speculatively) modeled by embedding the feedforward model into a larger system.

### 1.3.2 Hierarchical architecture

The architecture of HMAX is very similar to that of convolutional networks (CNNs) [23, 22], with the main differences being HMAX’s extra scale dimension $\lambda$ (§1.3.4) and the manner of training. One HMAX model is illustrated in fig.1-2. In general:

- An HMAX model has a series of layers roughly analogous to cortical areas along the ventral stream (V1, V2, V4, ....) However, each cortical area is modeled by a pair of HMAX layers: a layer of simple ($S$) units followed by one of complex ($C$) units. The overall sequence of layers in a model is thus $S_1, C_1, S_2, C_2, ....$

- Each layer consists of units (corresponding to cortical cells) which can be conceptualized as points sampling a 4D space $(x, y, \lambda, f)$.
  - Dimensions $x$ and $y$ correspond to position in the input image relative to the image center.
- The $\lambda$ dimension corresponds to scale or spatial wavelength ($1 / \text{spatial frequency}$) (§1.3.4).

- The $f$ dimension enumerates a set of features $F_i$ which is different for each pair of layers $(S_i, C_i)$. Within each pair, however, $f$ enumerates the same set of features in both the $S$ and $C$ layers.

- The input image is a 2D intensity image. HMAX disregards color and binocularity.

- A simple layer $S_i$ essentially convolves the preceding layer with a set of stored features $F_i$.
  
  - In $S_1$, each simple unit gets input from a local $(x,y)$ region of the input image.
  
  - In $S_i (i > 1)$, each simple unit gets input from a local $(x, y, \lambda)$ region of complex layer $C_{i-1}$ spanning all values of $f \in F_{i-1}$.

  - A unit’s response is the scalar output of some vector operation, such as normalized dot product (§1.3.3), between its input and a particular stored feature in $F_i$.

  - The overall effect is that the outputs of the preceding layer have been locally projected onto the new feature basis $F_i$.

- A complex layer $C_i$ locally pools the outputs of preceding layer $S_i$ units that encode the same feature $f \in F_i$.

  - Each complex unit gets input from a local $(x, y, \lambda)$ region of $S_i$ for a single value of $f$.

  - A unit’s response is the scalar output of a pooling function (e.g., max) over its inputs. It remains tuned to the same feature $f$ but is invariant over a wider range of $(x, y, \lambda)$.

  - This increased invariance allows subsampling in $(x, y, \lambda)$; the size of these
dimensions typically decreases from $S_i$ to $C_i$. By the topmost layer, they are very coarsely sampled, possibly down to the minimum of a single sample (via *global* pooling).

- For a single feedforward pass (through a trained model):
  - A new input image is loaded.
  - $S_1$ is computed from the input image, then subsequent layers are computed in sequence to the top.
  - The population activity of top-layer units forms the high-dimensional *signature* of the input image. This is converted to a vector and fed into a trained classifier to obtain a label.

- In training:
  - The number and sizes of layers are fixed parameters, as are the $(x, y, \lambda)$ receptive field sizes for units in each layer.
  - Feature set $F_1$ used in layer pair $(S_1, C_1)$ is also fixed (§1.3.4).
  - Feature sets $F_2, F_3, ...$ used in layer pairs $(S_2, C_2), (S_3, C_3), ...$ are learned. The simplest method is to sample features from training images. An element of $F_i$ may be learned from an input image by computing up to $C_{i-1}$ and then extracting the $C_{i-1}$ outputs that would be the inputs to some single $S_i$ unit.
  - The final classifier is trained using the top-layer signatures of labeled images.

### 1.3.3 Normalized dot product

A typical $S$ layer response function is *normalized dot product*: the dot product between two normalized vectors, one representing a stored feature and the other a unit’s
current input. The normalized version of a vector $\vec{v}$ is:

$$\frac{\vec{v} - \hat{\vec{v}}}{\max(\|\vec{v} - \hat{\vec{v}}\|, c\sqrt{D}/2)}$$

where:

- $\vec{v}$ is a $D$-dimensional vector with each $v_i \in [0, 1]$
- $\hat{\vec{v}} = \sum v_i / D$ is the mean of the elements of $\vec{v}$
- $c \in (0, 1]$ is a regularizing parameter which prevents excessive amplification of weak inputs
- $\sqrt{D}/2$ is the maximum possible value of $\|\vec{v} - \hat{\vec{v}}\|$

Normalization provides invariance to additive and multiplicative changes in the input; in layer $S_1$ the latter is contrast invariance.

Parameter $c$ provides a necessary limit to contrast invariance. Without it, every vector would be normalized to length 1, which is the same as saying all inputs are considered to have the same power. But the variance in a very weak input, such as a near-constant pixel patch for $S_1$, may be mostly due to noise. $c$ defines an upper bound on the amplification of weak inputs. For $S_1$, to prevent the model from doing things like hallucinating edges in blank walls, we set $c = 0.15$. As a result:

- An absolutely constant input is normalized to $\vec{0}$.
- Weak inputs having $\|\vec{v} - \hat{\vec{v}}\|$ between 0% and 15% of maximum are normalized to lengths from 0 to 1.
- All other inputs are normalized to length 1.

Cosine similarity is a similar measure, but it lacks mean subtraction and regularization parameter $c$.

The scalar result of a normalized dot product falls within $[-1, 1]$. Negative outputs must be rectified somewhere before the next $S$ layer that uses normalized dot product.

### 1.3.4 Scale dimension

HMAX's scale dimension $\lambda$ arises in layer $S_1$, which is special.
• Its feature set $F_1$ is a set of oriented 2D Gabor filters which uniformly sample orientation ([12] suggests at least 16-20 sample points.) Gabor filters are a good model of the optimal stimuli of V1 simple cells. The product of a sinusoid with a gaussian envelope, a Gabor filter is localized in space and scale and tuned to a particular orientation. Fixed parameters $\lambda_0$ and $\sigma_0$ define a set of filters having bandpass centered at spatial wavelength $\lambda_0$.

• Each oriented filter is convolved with the input image not once, but many times, each time resized by a different scale factor. $\lambda_0$ and $\sigma_0$ define the smallest set of filters. Each scale factor $s \geq 1$ yields a different scale band in $S_1$ sensitive to a range of spatial wavelengths centered at $\lambda = \lambda_0 s$. $s$ is logarithmically sampled, so adjacent bands are a fixed number of octaves apart, e.g., $1/4$ oct. (Pollen and Feldon [37] found $1/2$ octave sampling in cat visual cortex, while Silverman et al. [46] found $1/3$ octave for monkeys.)

• Within each scale band, $x$ and $y$ are uniformly sampled with interval $\propto \lambda$. This means smaller bands (having smaller $\lambda$) require more units to cover the input image. This is sketched in fig. 1-2. A typical sampling interval is $\lambda/2$.

Once the $\lambda$ dimension has been generated in $S_1$, subsequent layers can treat it much like $x$ and $y$.

• The representation in an HMAX (or CNN) layer is covariant to translation: shifting in the $(x, y)$ plane by an object in the input image induces a corresponding shift along those dimensions in every layer. Thus, invariance to translation can be increased by computing the same local feature at every $(x, y)$ (in an $S$ layer) and locally pooling the results (in the subsequent $C$ layer).

• With their extra $\lambda$ dimension, HMAX layers are covariant under the transformation of scaling as well.\(^1\) As an object in the input image grows larger or smaller, shifting is induced along $\lambda$ in every layer. HMAX achieves invariance to scaling in the same way as translation. $\lambda$ can be seen as almost a third spatial dimension.

\(^1\)As long as $C$ unit pooling ranges remain the same for each band, in terms of number of units pooled.
dimension, a proxy for size or depth.

This explicit representation of scale is a key difference between HMAX and CNNs, which have no built-in mechanism to handle scale invariance.
Chapter 2

i-theory

i-theory is an effort led by T. Poggio [35] to explain the good performance of hierarchical models in terms of a formal theory of invariance to transformations. Its starting point is the observation that these models continue to perform well when their simple cells are tuned to features unrelated to the objects being categorized; see fig. 2-1.

2.1 General invariance via projection onto transformed templates

Given the remarkable similarity among different cortical areas, a strongly-held hope among computational neuroscientists who study the ventral stream is that its operation can be described recursively, i.e., as the repeated application, by each successive area, of the same algorithms for learning and inference to the outputs of the previous area. Examples of recursive models inspired by the ventral stream include the Neocognitron, convolutional networks, and HMAX, all of which involve repeated iterations of the simple cell - complex cell concept introduced by Hubel and Wiesel.

The role of simple cells is typically framed as the building up of selectivity, with complex cells building up invariance to translation. Both these properties are known to increase along the ventral stream hierarchy. Under i-theory, invariance takes center stage. Viewed as a population, the original input pixels are already highly distinctive.
Figure 2-1: performance of an HMAX model similar to that in fig. 1-2 having $S_2$ templates sampled from training images, unrelated natural images, and random dot images on the Caltech 101 categorization task. Inset: an example random dot image.
i-theory sees the role of the ventral stream as increasing invariance to transformations while maintaining distinctiveness. It provides simple and complex cells with specific computational roles.

We propose that in a complex layer, the job of a column of cells (a set of cells which all have the same receptive field) is to compute a distinctive signature for the stimulus within its receptive field which is invariant to some set of transformations, even if the particular stimulus has never been seen before. A column of complex cells, together with the simple cells they pool, can accomplish this using an augmented random projection algorithm.

The classic result of Johnson and Lindenstrauss says, informally, that any set of $n$ points in $d$-dimensional Euclidean space can be embedded into $k$-dimensional Euclidean space via projection onto $k$ random vectors $t_1, t_2, \ldots, t_k$, where $k$ is logarithmic in $n$ and independent of $d$, so that all pairwise distances are maintained within an arbitrarily small factor.

Now consider a transformation acting on an image. (For now, we ignore the issue of limited receptive field sizes, and we assume the transformation can be discretized into a finite number of steps.) Random projections alone do not help us here – as the image changes, so does its projection. But now:

- replace each vector (template) $t_i$ with a set: all transformed versions of $t_i$, and
- replace the (scalar) result of projection onto $t_i$ with some order-invariant (scalar) statistic of the results of projection onto each transformed $t_i$ (such as sum, max, a moment or a histogram bin).

It can be shown [35] that this produces a signature invariant to the transformation but still distinctive for the input. Intuitively: to learn invariant recognition for an object, we do not need to see all transformed versions of it, as long as we are projecting onto a set of templates for which we have seen all transformed versions. Mapping this hypothesis onto convolutional architectures: simple cells perform the projection of the input onto a set of transformed templates, and a complex cell computes an order-invariant statistic (sum, max, etc.) over the set of simple cell outputs.
Importantly, the templates need not be related to the object being categorized. What is required is that transformation $g$ have the property that, for image $I$ and stored template $t$:

$$< g(I), t > = < I, g^{-1}(t) >$$ \hspace{1cm} (2.1)

Intuitively, this says that transforming $I$ has the same effect on the dot product as inverse-transforming $t$.

Formal proofs for affine transformations, based on an extension of the Johnson/Lindenstrauss random projection lemma, are given in [35].\(^1\)

### 2.2 Efficient computation of invariant signatures using learned energy-based features

Depending on the transformation, a direct application of the modified Johnson-Lindenstrauss principle above could require a large number of transformed versions of each template. Translation, for example, has two degrees of freedom, translation in $x$ and in $y$. If each of these is discretized into $m$ steps then we would require $m^2$ transformed versions of each template, i.e., $m^2$ simple cells per complex cell. When there are more transformation parameters, this becomes intractable. Moreover, for area V1, we know that simple cells are well-modeled by Gabor filters – not random templates.

In the following for clarity we will refer to our model of V1 as L1 (level 1). Simple cells in L1 are $S_1$ cells, and complex cells in L1 are $C_1$ cells.

#### 2.2.1 Gabor filter responses as components of a translation-invariant signature

The equation for a 2-d Gabor filter having orientation $\theta$ and phase $\phi$ is:

\(^1\)The formal proofs are primarily the work of F. Anselmi and T. Poggio.
\[ f_{\theta,\phi}(u, v) = \exp \left( -\frac{u^2 + v^2}{2\sigma^2} \right) \cos \left( \frac{360}{\lambda} u + \phi \right) \]

where \( u = x \cos \theta - y \sin \theta \), \( v = x \sin \theta + y \cos \theta \), \( x \) and \( y \) are coordinates within the filter, \( \sigma \) determines the width of the gaussian envelope, and \( \lambda \) is the wavelength of the sinusoid. The gaussian envelope represents the limited receptive field (RF) of an \( S_1 \) cell.

It turns out that if our templates are Gabor filters, then by assuming an energy model \([1]\) for complex cells, we can reduce the number of simple cells per complex cell from \( m^2 \) to 2. Under the energy model, each \( C_1 \) cell computes its output for an image patch \( P \) by squaring and summing the outputs of two simple cells which are tuned to Gabor filters having the same orientation but 90° out of phase (a ‘quadrature pair’):

\[ C_\theta(P) = \sqrt{S_{\theta,0}(P)^2 + S_{\theta,90}(P)^2} \quad (2.2) \]

A model column consists of all cells in a level that share the same receptive field. The different complex cells in an L1 column each compute a different measurement: \( C_0 \) measures the amount of vertically oriented energy (at frequency \( 1/\lambda \)) in the input patch, \( C_{90} \) the amount of horizontally oriented energy, etc. Together these cells form a \( k \)-dimensional signature for the content of the column’s receptive field. But how invariant is this signature to translations?

By ‘invariance’ we mean that as a stimulus within a column’s receptive field undergoes translation, the responses of the column’s complex cells remain the same relative to the gaussian envelope of their constituent Gabor filters. Note that this definition specifies what must happen in the limited case of (a) an input image which is blank except for the stimulus, which (b) is small enough to fit entirely within the column’s receptive field, and (c) remains inside the receptive field during the translation in question. In the more general case, when image content enters and leaves the receptive field, cell responses are expected to change accordingly.

Now consider one complex cell, for example, \( C_0 \) – the one having vertical orientation.
• It is easy to see that this cell will be invariant to vertical translation of a stimulus, since its constituent Gabor filters are – relative to the gaussian envelope – identical at every vertical position.

• Along the horizontal direction, the complex cell’s two simple cells essentially perform a Fourier transform for the single spatial frequency $1/\lambda$. Equation 2.2 then computes the phase-invariant magnitude of that frequency component. This gives us invariance to horizontal translation.

Now, since translation in any direction is separable into horizontal and vertical components, our example complex cell $C_0$ will be invariant to all translations. The same argument applies to every complex cell $C_0$. We therefore have a $k$-dimensional signature for the contents of an L1 receptive field that is locally invariant to translation in any direction.

Fig. 2-2 shows the invariant response of a single $C_1$ cell, and of a column of $C_1$ cells, to a stimulus translating across the receptive field.

2.2.2 Learning algorithm

A quadrature pair of simple cells for one $C_1$ cell can be learned via a simple PCA-based algorithm on video frames showing translation in a single direction, viewed through a gaussian aperture. (Extracting top eigenvectors has a neurally plausible implementation using the Oja method [31].)

The algorithm is inspired by the fact that the eigenvectors of a circulant matrix are complex exponentials. The rows of a circulant matrix can be seen as frames of a 1-d video of translation. Simply finding the top two eigenvectors yields a quadrature pair. Imposing a gaussian envelope yields 1-d Gabor filters. A key point is that it does not matter what pattern is translating – although this may affect $\lambda$.

For the 2-d case a little extra is required. For horizontal translation (for example), each row of the resulting template is still a sinusoid, but the different rows are not necessarily in phase; see fig. 2-3. Adding a temporal derivative, and using frames from multiple images (both consistent with biology) yield the expected result; see fig. 2-4.
Figure 2-2: Top left: a stimulus, and one (sine) Gabor filter across whose receptive field the stimulus is translating. Bottom left: the response of a single $C_1$ cell (the one representing vertical orientation) and its constituent $S_1$ cells to the translating stimulus. Right: illustrates the invariant response of a column of $C_1$ cells (8 cells of different orientations) to a translating stimulus. At each position, the $C_1$ vector’s normalized correlation to that of the untranslated stimulus is plotted, along with the same correlation value for other cell populations – $S_1$ cells and input pixels. Note that the $C_1$ curve remains flat for some time. This is due to normalized correlation – while the absolute response of each $C_1$ cell drops immediately due to the gaussian envelope, they all do so at the same rate, and thus the direction of the $C_1$ vector does not change.
More formally, to train one complex cell from $M$ example sequences of one specific transformation (translation in direction $\theta$), assuming the size of the complex cell’s receptive field (input image patch) is $[n_i \times n_i]$:

1. For each of $M$ image sequences:
   - Apply a difference-of-gaussians (DoG) spatial filter to each frame. This mimics the center-surround filtering of the retina and limits the range of spatial frequencies in the input.
   - Extract the image patch for each frame, generating an array of size $[n_i \times n_i \times n_f]$, where $n_f$ is the number of frames.
   - Take the temporal derivative. This leaves an array of size $[n_i \times n_i \times n_f - 1]$.
   - Apply an $[n_i \times n_i]$ gaussian mask to each frame. This will determine the gaussian envelope of the resulting filters.

2. Concatenate the results of the previous step along the temporal dimension. We now have an $[n_i \times n_i \times n_d]$ array, where $n_d$ is the total number of ‘derivative-frames’ from all $M$ image sequences.

3. Perform principal component analysis (PCA) on the results, treating the $[n_i \times n_i \times n_d]$ array as $n_d$ points in an $n_i^2$-dimensional space.

4. The filters for the complex cell’s two simple cells are the top two $[n_i \times n_i]$ principal components (PCs).

The algorithm consistently finds two gaussian-modulated filters having waves consisting of (a) a sinusoidal component oscillating along the direction of optical flow in the image patch and (b) a constant component in the direction orthogonal to that of the sinusoid. The sinusoids of the two filters will be $90^\circ$ out of phase.

The temporal derivative induces an appropriate dependency on the temporal ordering of frames; phase synchronization then results from temporally correlated changes that occur due to spatially correlated pixels (image structure) passing across the receptive field. The resulting algorithm is essentially the opposite of slow feature
analysis [51]: we find the components along which the input is changing the \textit{fastest} – and pool over them to produce invariance. The algorithm learns Gabor filters even when random dot or noise images are used\textsuperscript{2}. Unlike SFA, it yields tuning closer to real V1 cells, and is not susceptible to learning trivially invariant features, without additional assumptions. These properties arise from a straightforward application of a general theory (i-theory) to the case of V1.

Note that we assume the gaussian aperture is given prior to learning. Since V1 contains cells of varying receptive field size, we therefore assume the prior existence of multiple, overlaid ‘lattices’ of columns in V1, with each lattice corresponding to a different receptive field size.

\textsuperscript{2}This independence from the statistics of the moving training images suggests significant prenatal learning of V1 simple cells could occur via spontaneous retinal waves.
2.2.3 Limitations

PCA requires that all the components extracted must be mutually orthogonal. This has two consequences for us.

First, after the initial quadrature pair, we do not obtain additional pairs for different spatial frequencies. Why not? While two sinusoids of different frequencies are orthogonal over an infinite domain, they are not orthogonal when restricted to a small gaussian aperture. Thus, PCA cannot output them. This is actually beneficial – as mentioned above, to match physiology we want $\sigma \propto \lambda$, so we do not want cells in the same column to learn different $\lambda$.

Second, we cannot learn all the complex cells in a column at once via a single PCA over examples of translation in all directions. The reason is similar to that excluding multiple frequencies: within a small gaussian aperture, 2-d sinusoids of different orientation are not orthogonal (unless the orientations are separated by 90°). Therefore we can only use PCA to learn one complex cell at a time – one quadrature pair of a single orientation, for a single direction of translation. A fully
unsupervised algorithm for an entire column would still require a constraint that forces different complex cells to learn different orientations, but it must be weaker than an orthogonality constraint. We do not currently model this.

2.2.4 Other transformations

The property of a column of complex cells described in §2.2.1 – each learned from translation in a different direction, tuned to energy oriented orthogonally to that direction, but invariant to translation in all directions – could also be achieved for transformations other than translation. It is achievable for translation because every translation consists of some mixture of translation in x and translation in y, two component transformations which produce orthogonal motion at every point in the receptive field. We believe V1 computes invariance only to translation because translation is all a V1 column is likely to see through its small aperture. But centered rotation and scaling, for example, also produce orthogonal motion vectors. One could imagine a column of complex cells, each tuned to a feature representing some combination of centered rotation and scaling, but invariant to any such combination. (In this case, the simple cell templates could not strictly be called ‘Gabor’ filters.) Fig. 2-5 and fig. 2-6 show templates learned from pure (centered) rotation and scaling, respectively.

It should be noted that such \( C_1 \) cells, tuned to combinations of rotation and scaling, would not at the same time be translation invariant. In the sections that follow, we restrict the set of potential templates to those learnable from (approximately) pure shifts in the input. This ensures that the entire population of learned templates will have the same invariance properties\(^3\). In L1 this restricts us to translations, but as we will see, there are more degrees of freedom above L1.

\(^3\)This could probably be enforced during learning by penalizing instability over small times scales
Figure 2-5: $S_1$ templates learned from observing centered rotation. Each row is a different gaussian aperture size, shown in the first column. Remaining columns are the first six principal components.
Figure 2-6: $S_1$ templates learned from observing centered expansion. Each row is a
different gaussian aperture size, shown in the first column. Remaining columns are the
first two principal components.
2.2.5 Matching V1 tuning

Regardless of RF size, the number of cycles, i.e., the value $\sigma/\lambda$, seems to be fairly constant as well as consistent over several species. With some additional plausible assumptions with respect to processing of the retinal image before V1, our algorithm reproduces this relationship.

Solving the eigenvalue equation and assuming the $1/f^2$ power spectrum of natural images yields the same relationship analytically, including the compression of the gaussian envelope along the modulated axis which is seen for real cells. The constraint on $\sigma/\lambda$ relies on the persistence of a DC component after retinal preprocessing and on the orthogonality constraint of PCA. The DC component remains after preprocessing due to an ‘imperfect’ derivative function which passes some small fraction of the DC; see [35] for details.

2.3 Extending the energy-based model to higher levels

While the inputs to L1 were two-dimensional, its outputs are four-dimensional. Because L1 receptive field sizes were limited, the dimensions $x$ and $y$ still exist in L1’s output (although resolution has been reduced). However, at each $(x, y)$ position we now have many complex cells, each computed from simple cells having different orientations $\theta$ and wavelengths $\lambda$.

We have performed several preliminary experiments in this space of L1 outputs. For simplicity we have omitted scale, leaving three dimensions: $x$, $y$, and $\theta$.

2.3.1 Learning in $C_1$ space

The learning algorithm of §2.2.2 can be applied to L1 outputs without modification. Fig.2-8 shows the results for translation. As one might expect, the result is the same Gabor filters we would learn in pixel space, but replicated across each L1 orientation plane. Also in fig.2-8 we show that a column of $C_2$ cells learned this way is translation
Figure 2-7: Gabor filter shapes learned by our algorithm, overlaying shapes found for macaques [42], cats [18], and mice [30].
invariant. Due to convergence, and since L1 has already downsampled the image in $(x, y)$, an L2 column has a larger receptive field.

In fig. 2-9 we show the results of a similar experiment with a different transformation: off-center rotation. The image rotated about a center at the edge of the L2 column's receptive field. The result is Gabor-like templates that are oriented not only in the $(x, y)$ plane but also with respect to $\theta$. A complex cell built from templates like these has a different selectivity than those learned from pure translation. But as long as each 2-d slice is approximately a Gabor filter, it will be approximately translation invariant.

2.3.2 Implications of higher dimensionality

The off-center rotation experiment (fig. 2-9) illuminates the fact that there are more degrees of freedom in $C_1$ space than in pixel space.

In 2-d pixel space, the set of templates learnable from pure shifts can be enumerated by a single parameter, $\theta$ (again, we are ignoring the wavelength $\lambda$ for now). $\theta$ then becomes a dimension of our 3-d L1 output space. Now, in this 3-d space, the set of pure shifts is larger, requiring two parameters to enumerate, and thus potentially
many more templates could be learned.

However, arbitrary shifts in $C_1$ space are not permitted: they must arise from transformations in the original image. For example, given that our 3rd dimension is $\theta$, we might expect that rotations in pixel space would correspond to shifts along $\theta$ in $C_1$ space – but this is actually only true in the limiting case of a very small object. Rotation of a larger object will produce shifting along $\theta$ but at the same time, rotation in $x$ and $y$; overall this is not a pure shift.

What kind of rotation invariance would such $S_2$ templates yield? Has anything been gained? Actually, something has. Fig. 2-10 shows that rotations are closer to being pure shifts in $C_1$ space than in pixel space – and therefore, L2 will be more invariant to them than L1 is.

### 2.3.3 Visualizing higher-order features

While Gabor-like $C_2$ features are simply described in $C_1$ space, it is difficult to predict what they will look like in pixel space. In fig. 2-11 we probe two theoretical $C_2$ features with stimuli used in [6] to test V4 neurons. Note we do not claim to fit that paper’s results; we only note that it is possible to produce a wide variety of selectivity patterns by varying only the orientation of a Gabor-like filter in $C_1$ space.
**Figure 2-10:** Experiment showing that rotations are more shift-like in $C_1$ space than in pixel space. Top left: one frame of a stimulus undergoing rotation. Top right: one frame of the $C_1$ layer response to the rotating stimulus (one orientation plane shown). Bottom left: one frame of the rotating stimulus, blurred and downscaled to the same spatial resolution as $C_1$. Bottom right: normalized correlation between the rotated stimulus and a shifted (in 3-d) version of the original for different cell populations and different amounts of rotation. The rotation is more easily represented as a shift in $C_1$ space than in pixel space. Note that simple blurring outperforms both but loses distinctiveness.
Figure 2-11: The responses of two theoretical $C_2$ cells to stimuli used to test V4 neurons in [6]. Top: the $S_2$ units' sinusoid is parallel to the $x$ axis. Bottom: the sinusoid is parallel to the $\theta$ axis.

2.4 Covariant representations

For i-theory to apply to hierarchical models, we must be certain that the conditions required by i-theory hold for levels above the first. Could we, for example, learn invariance to rotation in $S_2$ using the translation-invariant outputs of $C_1$? This requires $C_1$ outputs to be covariant under rotation, which means that the original i-theory assumption (equation 2.1) continues to hold in $C_1$ output space. If $C_1(I)$ is the output of layer $C_1$ for image $I$, and $h$ is rotation, we now require:

$$< C_1(h(I)), C_1(t) > = < C_1(I), C_1(h^{-1}(t)) >$$

(2.3)

Fig. 2-12 shows that $C_1$ is covariant under rotation only when the full set of Gabor filter orientations is present. Intuitively, if $S_1$’s templates adequately sample all orientations, as the image rotates, $C_1$ activity will shift along the feature dimension, from one orientation to the next. When some orientations are removed, this is no longer the case.

This result would seem quite limiting for i-theory, as it suggests lower levels have
to anticipate the invariances higher levels will learn. However, fig. 2-13 shows that covariance under rotation does approximately hold for $C_2$, despite $S_2$ templates being random image patches, each sampled at only one rotation. We postulate that the much higher number of $S_2$ templates approximates the ‘full coverage’ property that a full $S_1$ possesses.
2.5 Summary of contributions

- We interpret V1 simple and complex cells in the context of a new theory of invariance. Under this interpretation, sets of Gabor filters are not derived from image statistics, but instead arise due to their suitability for economically computing a high-dimensional, locally translation-invariant signature for the contents of a V1 receptive field.

- We describe a simple algorithm by which a model complex cell – in L1 or a higher level – can learn Gabor-like templates suitable for computing one component of a transformation-invariant signature.

- We obtain a good fit to V1 data from several animal species.

- We recap preliminary investigations – conceptual and experimental – into the extension of these principles into a recursive hierarchical model.

- We demonstrate that a precondition of i-theory, that of covariance, can hold in upper layers, even for transformations not anticipated in the training of lower layers.

Acknowledgements

The i-theory project was conceived by T. Poggio and involved many lab members. T. Poggio, Fabio Anselmi, Andrea Tacchetti, Joel Leibo, and Lorenzo Rosasco participated in many theoretical discussions whose topics included most of the work in this chapter. J. Leibo performed some early experiments learning Gabor filters from translating images, and A. Tacchetti was a collaborator in fitting learned filters to animal V1 data. Formal proofs referenced were the work of F. Anselmi and T. Poggio.
Chapter 3

Scale-invariant object categorization under eccentricity-dependent retinal resolution

Feedforward hierarchical models of primate object recognition ignore the nonuniform resolution of the retina, for which the smallest perceptible spatial wavelength $\lambda$ is roughly proportional to eccentricity from the fixation point. Incorporating this detail forces us to clarify the scope of such models – i.e., what problem is solved by a single pass through the model – and to reevaluate aspects of the models themselves, especially the role of scale invariance. We present theoretical and experimental findings for scale-invariant object categorization in the context of an HMAX model modified to allow for realistic retinal resolution.

Our experiments explored two aspects of this model related to the handling of scale: (1) the size and shape of the input window used to select a subset of the visual information available in a scene for processing in a single feedforward pass, defined as a region in $(x, y, \lambda)$, and (2) the handling of the $\lambda$ dimension within the hierarchy. We also investigated the model’s effectiveness in the presence of clutter.

Our main experimental results are (1) spatial wavelengths too small for the retina to perceive across the entire object do not play a significant role in the no-clutter case,
but confer robustness in the presence of clutter, and (2) preservation by the hierarchy of information about the relative scale (distance along $\lambda$) of feature activations is more important than current models reflect.

3.1 Theory

As a model of primate vision, HMAX as described in §1.3 fails to account for a fundamental fact: resolution across the primate retina is not constant.

While retinal resolution imposes limitations on primate vision, mainly in the periphery, the limitations themselves are not our focus. By working within the same constraints, we can study the choices evolution has made. Our goals are to better understand both the ventral stream and the visual problem of categorization itself, and to use this knowledge to improve computer vision systems.

Implementing nonuniform resolution in a computational model is straightforward; however, incorporating it into our understanding of the model forces us to:

- Clarify the problem definition: what is accomplished in a single feedforward pass?
- Reevaluate the roles of different parts of the model.

We provide these theoretical results in this section. As they represent a fairly broad rethinking of HMAX and similar models, not every claim is tested in this study. We indicate untested claims where appropriate.

3.1.1 Object recognition

We begin by clarifying problem scope and terminology.

3.1.1.1 Objects and labels

For our purposes, an object is some physical thing that occupies a connected region of 3d space. A discussion of which such things 'should' qualify as objects, and where boundaries should be drawn between objects, is beyond our present scope. We simply
take it as given that an organism or agent needs to be able to associate certain objects with *labels*.

A label can denote a specific object or a class of objects. For a human or artificial visual system, labels can have names, but animals demonstrate nonverbal labeling when they behave appropriately to objects they encounter. The label, however represented, indicates something semantically meaningful.

How the brain represents labels is also outside our scope. We model labels as a set of discrete symbols. Typically the set is given, along with a set of labeled examples for supervised learning.

The same object can have multiple labels. One reason for this is different levels of generalization. For example, the same object could have labels ‘Tomaso’, ‘human’, ‘male human’, etc. We do not explicitly model these ‘is-a’ relationships between labels. Also, since our scope is limited to the visual system, labels representing abstract categories (e.g., mammals, vertebrates) that cannot be directly inferred from the visual appearance of the object are disallowed.

Many objects can move independently of other objects in a scene, but many (e.g., houses, trees) cannot. Many objects have closed boundaries, but some (e.g., roads, rivers) effectively do not.

A *categorization task* is a particular mapping from images to a set of labels. The kind of task depends on the labels:

- **basic categorization**: labels are basic object categories (cat, dog, human, car, ...)

- **individual recognition**: labels identify particular instances of a category (e.g., different people)

- **object properties** (e.g., sad face, happy face, angry face, ...)

### 3.1.1.2 Hierarchies of objects and parts

A *part* is an object that is an attached component of a larger object. For example, a nose is part of a face. Many objects that we wish to label are parts of other objects.
Object-part is a 'has-a' relationship. It may be probabilistic: a part may be detachable or, for categorization, the relationship may not hold for all instances of the object or part. In these cases, the presence of an object is only informative for the presence of a part, and vice-versa. Note that even when an object has some part, the part may not be visible in the image due to occlusion by the rest of the object or by an unrelated object.

Parts may in turn have their own parts, forming hierarchies. Animate objects tend to be at the top of such hierarchies, since the presence of such an object may not be very informative for any larger object, or vice-versa. Nevertheless, for example, roads and cars might be seen together so frequently that they could be considered to have a high-probability object-part relationship.

3.1.1.3 Parts vs. features

A part is not the same thing as a visual feature. Parts are full-fledged objects. In terms of physical construction, an object is by definition more complex than a part, but on average, objects and parts have equal visual complexity. Consider the task of assigning category labels to objects seen in isolation in image patches of size $P \times P$ pixels. Suppose two of the many categories are 'car' and 'headlight'. If we start with very small $P$ and increase $P$ (and the resolution) until some fixed level of performance is reached, will the threshold $P$ be larger for cars or for headlights? In general, the answer to this question cannot be predicted from the direction of the object-part relationship.

Labels for objects, which may be parts of other objects, are visual system outputs. Visual features are used in intermediate computations within a visual system. They might represent the preferred inputs of various cells. A high value for some visual feature might be highly informative for some object, but features need not have any semantic meaning.
3.1.1.4 General transformations and invariance

Transformations act on objects in ways that change their appearance in the image but preserve labels. A visual system's invariance to transformations refers to its ability to maintain some level of performance of a task while one or more objects in the scene are transformed.

Translation, scaling, and rotation constitute the rigid transformations. Under planar rotation the object continues to show the same face to the viewer; for rotation in depth it does not.

When transformations are easily parameterized, as are the rigid transformations, we can speak of an invariance range – the range of the transformation parameter within which task performance remains above some level.

Many kinds of objects can undergo deformation, or nonrigid transformations. These may be approximated as locally rigid.

Clutter can be seen as a transformation, although it does not act directly on an object, but on the surrounding scene. Clutter could even partially occlude an object from the viewer.

Different kinds of objects may be subject to other forms of variability which can be considered transformations. For example, people wear different clothes from day to day. Robust recognition of individual people requires invariance to this transformation.

Change of identity, or intra-class variability, may be seen as a transformation for categorical labels. For a visual system to correctly output the label 'car', it must be invariant to which particular car is present. For some categories this might be equivalent to invariance to deformations. For labels denoting individual objects, however, invariance to deformations is desirable while invariance to change of identity is not.
Figure 3-1: upper left: a uniform window (§3.1.2.1) shown in $(x, \lambda)$ cross-section. A $(y, \lambda)$ cross-section would be identical. In a uniform window, every scale band spans the same $(x, y)$ region. The sampling interval in $(x, y)$ is $\propto \lambda$ for each band. Blue dots illustrate relative sampling density among scale bands; the actual number of bands and samples may be different. Upper right: a retinal window (§3.1.2.2). The lighter-shaded regions marked with ‘X’ represent information that is ‘missing’ (relative to a uniform window) due to eccentricity-dependent resolution. Lower: the same two windows shown with all spacing equalized to emphasize the relative number of samples in different scale bands.
3.1.2 Capture and input windows

In existing HMAX the input image is just a 2D patch of pixels at uniform resolution (see fig. 1-2). Where that patch came from is outside the scope of HMAX; it could be an entire image or a patch extracted from a 2D window which was placed within a larger image by some parent process.

To think about nonuniform resolution, we need to augment the window concept to also include $\lambda$. We define an input window to be a 3D region in $(x, y, \lambda)$ identifying a particular subset of the visual information in a scene for processing in a single feedforward pass, resulting in a signature.

A vision system places an input window within a scene via a combination of physical aiming of the capture device (camera or retina) and attentional masking. The capture device has a window of its own: the region of $(x, y, \lambda)$ it can ‘see’. This capture window is determined by the construction of the device and the direction in which it is currently aimed. The input window is a subset of the capture window, and attentional masking is simply the process of selecting only this subset for processing through the feedforward hierarchy. Physical re-aiming is only necessary when the desired input window does not fall entirely within the current capture window.

The capture window’s placement within the scene may change, but its size and shape in $(x, y, \lambda)$ do not. On the other hand, the size and shape of the input window could conceivably change for every feedforward pass, subject to the constraint of fitting inside a capture window. The appropriate size and shape of the input window for certain tasks is the major focus of this study.

3.1.2.1 Uniform window shape

A uniform window is just the existing HMAX window shape with $\lambda$ made explicit. It includes the same range of spatial frequencies at every position; see fig. 3-1 (left). The capture window of a digital camera is a uniform window.
3.1.2.2 Retinal window shape

The capture window of the primate retina is not uniform; its resolution is eccentricity-dependent. The smallest perceptible spatial wavelength is roughly proportional to eccentricity from the central fixation point. Adding the constraint of an absolute minimum wavelength $\lambda_0 > 0$ implies a small central patch of constant high resolution. The resulting retinal window is shaped as in fig. 3-1 (right). We call the small central region of highest resolution the foveola [36].

Every scale band in a retinal window spans the same number of wavelengths $N$ and thus requires the same number of samples. Rough estimates for the human retina are $N \approx 15-20$ cycles with a foveola diameter around $0.5^\circ$ of visual angle [36].

The retinal window shape was introduced in [36] where it is called an ‘inverted truncated pyramid’. It is a functionally-inspired model based on the observations of Anstis [2] regarding scale invariance. It differs from the descriptive model of Schwartz [43] in having a central region of highest resolution instead of a single point. Only a retinal window, with a foveola, can have a constant $N$ cycles at every $\lambda$.

We have implemented a version of HMAX that supports a retinal input window. Note that for our purposes, it isn’t necessary to simulate retinal resolution directly in the input image. We simply remove the $S_1$ units corresponding to the regions marked ‘X’ in fig. 3-1 (right), giving the $S_1$ layer a retinal shape in $(x, y, \lambda)$ which is then inherited by subsequent layers (sometimes with subsampling).

3.1.2.3 Attentional masking

To recognize an object which is small relative to the capture window, the entire contents of the capture window cannot be used to compute the object’s signature. To obtain an appropriate input window, some ability to suppress information outside an attentional region in $(x, y)$, or possibly $(x, y, \lambda)$, is needed.

As already noted, after masking, the resulting input window need not have the same shape as the capture window. For example, masking within a retinal capture window could yield a uniform input window, and vice-versa.
In this study, for the retina, we focus on masking centered at the fixation point. While some ability to recognize unfixated objects is clearly required for the planning of subsequent fixations, and sometimes for social reasons, the ‘core competency’ of the primate ventral stream is the generation of an invariant signature for an object being fixated. When we want to recognize something, we typically fixate it.

Computationally speaking, masking could be applied at any level of the hierarchy as long as global pooling has not yet occurred. The ability to mask with precision decreases with each $C$ step due to local pooling.

### 3.1.3 Problem scope

We consider object categorization in two scenarios:

- Isolated object on a blank background. (Note: most of our experiments consider this case.)
- Complex scenes.

In both scenarios our task is to answer questions of the form ‘what is this?’

- For complex scenes, ‘this’ is given as in §3.1.4.
- ‘What’ must be one of the labels used to train the final classifier.

Not wishing to tackle more general scene understanding, we restrict ourselves to answering each such question via either a single input window placement (one feed-forward pass), or at most, a small number of placements with some simple rule for choosing among the resulting labels.

### 3.1.4 Invariance to object position and scale

Invariance to position and scale is achieved in HMAX and CNNs via pooling of $S$ unit outputs by $C$ layers, iterated over several $(S, C)$ steps (§1.3.2). We must distinguish between two kinds of position/scale invariance:

- **Object-level** invariance to the position and size of the object of interest within the scene.
* Feature-level invariance to the relative separation of features in \((x, y, \lambda)\) within an object.

We focus here on object-level invariance; feature-level invariance plays a role in § 3.1.5.

In HMAX the importance of object-level invariance has likely been overemphasized. From a purely computational point of view, achieving object-level invariance in the isolated-object case is trivial, but in the complex-scene case, it can still play only a limited role. To ask ‘what is this’ for an object in a complex scene, ‘this’ must identify the location of the object. In an intelligent agent, ‘this’ would be the result of a prior decision step in a larger scene-understanding task, outside HMAX. Once the agent places the input window accordingly, the object-level invariance problem is already mostly solved; HMAX need only concern itself with object-level invariance to the degree that ‘this’ may be approximate.

The encoding of ‘this’ must not rely on the rest of the scene-understanding process already being completed. It must not be necessary to, for example, fully segment an object from background before trying to recognize it. Relying on a full object outline, or even a tight bounding box, requires too much work to have already been done. Segmentation and recognition are a pair of ‘chicken-or-egg’ problems common in vision: each can inform the other, and each becomes significantly easier if the other has already been solved.

We propose a point-size encoding for ‘this’: a point \((x, y)\) which is on the object, plus an approximate (apparent) diameter \(d\). Possibly \((x, y)\) could be restricted to within some distance of the object center. The object may be partially occluded but must be visible at \((x, y)\). Approximate size \(d\) is needed to resolve ambiguity between objects and parts. For example, if \((x, y)\) falls on a doorknob, is the desired answer ‘doorknob’, ‘door’, or ‘house’?

Any way of specifying ‘this’ which falls short of full segmentation implies that the input window will probably contain some clutter: information which is not informative for the object. HMAX must therefore tolerate some amount of it. The degree to

\[1\] At recognition time. However, training using fully segmented objects is acceptable and probably desirable.
which clutter can be tolerated constrains how accurate ‘this’ has to be. Deviation of \((\tilde{x}, \tilde{y})\) from the object center, and \(\tilde{d}\) from the true size, decrease both the fraction of the input window which contains object information and the fraction of all object information which is inside the window. If HMAX performance depends on these fractions remaining above some fixed limit, this implies that tolerances for \((\tilde{x}, \tilde{y})\) and \(\tilde{d}\) will be scale-invariant, i.e., expressible as a fixed proportion of apparent object size.

Note that while these observations were inspired by thinking about retinal resolution, they are independent of window shape or the specific mechanism of image capture.

Primate retinal resolution is a necessary biological tradeoff. Processing the entire visual field at the resolution of the foveola would expand the optic nerve and visual cortex far beyond any feasible size. Therefore, the \((\tilde{x}, \tilde{y})\) component of input window placement requires eye movement for best recognition performance. Near the fixation point, the retinal window spans a wide range of \(\lambda\), but only a fixed number of cycles \(N\) for each. As a fixated object moves closer or farther away, information shifts along the \(\lambda\) dimension without loss (except at the resolution limit). \(N\) cycles is presumably enough to allow for inaccuracies in \((\tilde{x}, \tilde{y})\) and for actually recognizing the object (§3.1.5). This broad, scale-invariant coverage near the fixation point means attention to a given size \(\tilde{d}\) can be accomplished by masking, i.e., without physical movement. That evolution would choose this scheme is not surprising: eye movements are much faster, cheaper and often safer than changing one’s distance from an object.

3.1.5 Invariance to within-object transformations

Assuming object-level invariance to translation and scaling, plus some tolerance to clutter, remaining label-preserving transformations include 3D rotation, deformation, and change of identity.\(^2\) These within-object transformations affect individual objects, not their relationship to the scene. The problem of discounting them is the same in both isolated-object and complex-scene scenarios. HMAX models have two

\(^2\)Change of identity (within a category) fits an operational definition of transformation as any change to an input image that should not change the output label.
Figure 3-2: illustrates a retinal window (§ 3.1.2.2) having $N$ cycles per band, centered on an unoccluded object of apparent diameter $d$. The central scale $\lambda_c = d/N$ is the wavelength at which the object is $N$ cycles across. Fine bands are the scale bands inside the window having $\lambda < \lambda_c$; these bands can only see part of the object. Coarse bands have $\lambda > \lambda_c$ and span more than the object. Note that for an object smaller than the foveola, the central scale will fall outside (below) the window ($\lambda_c < \lambda_0$), making all scale bands in the window coarse bands. Context regions contain information from the object’s background.
mechanisms for dealing with them:

- Invariance to the relative separation of features in \((x, y, \lambda)\) within an object.
- Training the final classifier using many examples for each label.

For some objects, invariance to 3D rotation almost certainly requires multiple training examples. The front and back views of an object (for example) may be very different.

Deformations and change of identity are difficult to formally characterize. At one extreme, a single \((S, C)\) layer pair which computes small, simple features and pools them over wide ranges of \((x, y, \lambda)\) would yield a ‘bag of features’ type representation having a high tolerance to deformation – so high that the resulting signatures are no longer reliably distinctive for the correct labels. A sequence of \((S, C)\) pairs which maintains a balance between feature complexity and how much those features can shift relative to each other seems to provide a better factorization of these difficult-to-characterize transformations [34, 40]. However, coming up with a good architecture remains an exercise in parameter-searching.

We make the observation that, at least at the level of recognizable parts within an object, relationships in scale are more stable than relationships in position. For example, a doorknob might appear on the left or right side of a door, but it never appears larger than the door frame.\(^3\) Relative size tends to vary less than relative position. This suggests that pooling along \(\lambda\) should be less aggressive than over \((x, y)\). We test this idea in §3.2.

### 3.1.6 Whole-part representation

Unless an object is small enough to fit entirely within the foveola, different points on the object will be seen at different resolutions. For an object centered perfectly in a retinal window, there will be scale bands covering less, more, and about exactly the entire object. Fig. 3-2 illustrates this and defines some terminology.

\(^3\) Under normal viewing conditions, excluding extreme closeups.
3.1.6.1 Fine bands

Fine bands see only part of the object, which would seem to be a disadvantage relative to the uniform window as less information is available. Another obvious drawback is that at recognition time we do not know which part will be fixated, so multiple fixations per example may be needed during training.

However, the uniform window also has its drawbacks. The corresponding bands in a uniform window would see the entire object, but due to their higher sampling rates, they have many more units (fig. 3-1, left), to the point of dominating the window. Bands with more units must be pooled more aggressively, so a balance of feature complexity and pooling (§ 3.1.5) that works for one band cannot be applied to others. Overly-aggressive pooling leads to loss of distinctiveness. We compare the performance of the two window shapes in § 3.2.

One way to sidestep the fine bands issue with the retina would be to use attentional masking to simply exclude fine bands from the input window. But there could be benefits to leaving them in:

- Greater distinctiveness due to retaining all the available object information, hopefully without loss of invariance. (Tested in § 3.2.)

- Better clutter tolerance. Fine bands are smaller in \((x, y)\) and some may be entirely free of clutter. (One test in § 3.3.)

- More robustness to partial occlusion. When we cannot see the entire object, fine bands will still contain a good representation of the visible part being fixated. (Not tested.)

3.1.6.2 Coarse bands

Since all coarse bands span the entire object, there is no obvious reason for masking in \(\lambda\) to exclude entire coarse bands. The attentional mask in \((x, y)\) should suffice. For bands that extend beyond it, the input window will be less than \(N\) cycles wide. The question is, how wide should the mask be in \((x, y)\)? Should the input window
intentionally include any information from the background? (See the regions labeled ‘context’ in fig. 3-2.)

For some object categories, particularly those which are parts of larger objects, contextual information could be highly informative. This could certainly help when the object of interest is small. (Not explored in this study.)

3.1.7 ‘Has-a’ relationships

Signatures derived from a retinal input window uniquely possess a property worth discussing here, though it is beyond the scope of this study’s experiments.

Consider a retinal input window centered on an object as in fig. 3-2. We’ll assume the isolated-object case for simplicity, but it is not critical. Assume pooling over \( \lambda \) in the hierarchy never becomes global, so the resulting signatures still have scale bands: signatures are 2D in \((\lambda, f)\).

Now, because every scale band in a retinal window is \( N \) cycles wide, processing in each band can be scale-invariant all the way to the signature. For each band, \( S \) layers apply the same sets of local features and \( C \) layers can use the same pooling ranges, including the global \((x, y)\) pooling step. This ensures that in the resulting \((\lambda, f)\) signature, the \( f \) dimension has the same meaning at every \( \lambda \). This cannot be the case for a different window shape, such as a uniform window, where \( N \) is not constant.

For an object larger than the foveola, there will be fine bands in the signature that cover only part of the object. Scale-invariant processing means that bands representing a part and bands representing an entire object are encoded in the same feature space. Under our paradigm, parts are of the same order of complexity as objects (undergoing identical processing), just smaller. This differs from the traditional interpretation of ‘part’ in HMAX and CNNs, where parts are assumed to be represented at levels of the hierarchy below the signature level.

This property could be exploited for tasks other than immediate object recognition. For example, a set of training signatures of this type could support an offline matching process to learn the hierarchical whole-part structure among the lexicon
of categories. The fact that fine bands of some ‘face’ signatures happen to match the coarse bands of ‘nose’ signatures (for example) implies a whole-part relationship between those two categories.

3.2 Experiments: isolated objects

This set of experiments focuses on scale-invariant object categorization in the isolated-object case. We train models using objects of one size, then test categorization performance for previously-unseen objects presented at a range of sizes.

This tests invariance to all the within-object transformations discussed in § 3.1.5 (and some of § 3.1.6) as well as invariance to object size. The latter is nontrivial even for an isolated object due to resolution limits: less information is available for a small object than a large one.

Not tested in these experiments are invariance to object position (central fixation is given for test objects), clutter, or occlusion.

Our focus is on the model’s handling of scale: the impact of a retinal vs. uniform input window, use of attentional masking in $\lambda$ to limit scale bands, and pooling over $\lambda$ in the hierarchy.

3.2.1 Model

We use an HMAX model similar to that in fig. 1-2, which is one of the best-performing HMAX models for object categorization [29].

- $S_1$ (retinal window): 12 oriented Gabor filters. $\sigma/\lambda = 0.5$, giving a half-amplitude bandwidth of $\approx 1$ octave [24]. $N = 15$ cycles sampled at $\lambda/2$. Scale bands span 3 octaves at 1/4 octave intervals. Response function is normalized dot product (§ 1.3.3). Note that all bands are circular in $(x, y)$.

- $S_1$ (uniform window): same, except that all scale bands cover the same $(x, y)$ area as the retinal window’s largest band.

- $C_1$: phase invariance is achieved using an energy model [1] over a quadrature
pair of Gabor filters.

- $S_2$: 1,024 features sampled from unrelated natural images. Features are 4x4x1 in $(x, y, \lambda)$. Response function is normalized dot product (§1.3.3) rectified using absolute value.

- $C_2$: max pooling, globally over $(x, y)$ and initially, globally over $\lambda$.

- **Classifier**: linear regularized least squares. Each dimension is preprocessed via $z$-scoring using training set statistics.

Since we are working with isolated objects, attentional masking in $(x, y)$ is unnecessary for this experiment. Masking in $\lambda$, if any, involves suppressing entire scale bands at the $S_1$ level.

Our initial model **M1** corresponds to traditional HMAX. It has a uniform window, global pooling over $\lambda$, and no $\lambda$ masking.

When pooling over $\lambda$ is no longer global, signatures must be aligned before they are input to the classifier as illustrated in fig.3-3. Since size $\tilde{d}$ is approximate during testing, classification is attempted using a range of $\tilde{d}$ and the highest-scoring category is chosen.
Figure 3-3: illustrates alignment along $\lambda$ necessary for signatures which have not been globally pooled over $\lambda$ (i.e., which still have scale bands) prior to their input into the classifier. This is done using the object’s known size $d$ (during training) or estimated size $\hat{d}$ (for recognition). After alignment, band 0 (red bar) is the central scale of the object, bands above the red bar are coarse bands, and bands below it are fine bands (see fig. 3-2), with the caveat that any local pooling over $\lambda$ will have blurred the bands somewhat. Due to resolution limits, signatures for different-sized objects may not fully overlap in $\lambda$; ‘missing’ bands are filled with z-score 0. Each blue box represents responses to the full set of top-level features, computed at a particular scale.

Note: explicit retention of scale bands at the signature stage and alignment via shifting is one algorithm for achieving the computational goal [26] of having a distinctive signature which preserves information on the relative scale of feature activations within an object without losing object-level invariance to size. While theoretically simple and easy to implement in a computer, we are not asserting its literal implementation in the ventral stream, which lacks an obvious mechanism for large-scale shifts of variable magnitude. The ventral stream might achieve the same goal using the equivalent of additional $(S, C)$ layers.
Figure 3-4: shows the range of object sizes used in the isolated-object experiments (§3.2). Size is stated in octaves relative to the foveola size.
3.2.2 Procedure

Images are from the Caltech 101 dataset. We use the full segmentation provided for the labeled object in each image to remove backgrounds. Images having multiple objects are discarded. The size of an object is taken to be the longer side of the object's bounding rectangle. Objects with more than a 2:1 ratio between the longer and shorter side are discarded. Categories having less than 16 images remaining are dropped. This leaves 95 categories. The training set has 15 images per category; remaining images (up to 30 per category) form the test set.

We evaluate categorization performance for every (train size, test size) condition, both over a 2.5 octave range at 1/4 octave intervals (see fig. 3-4). The smallest size is 1/2 the foveola (octave -1) and the largest is $2^{1.5} = 2.83$ times the foveola (octave 1.5). We train 11 classifiers, one for each training size, and test each classifier using objects of all sizes. Images are resized as needed using bicubic interpolation. For a given (train size, test size) condition, the score is the average of the per-category scores. The overall score for a model is the average score over all $11^2$ conditions.

3.2.3 Results

We present the results as an optimization path through model space, starting with initial model $M1$. In each of the following experiments, we start with the best performing model so far and attempt to improve on it by optimizing one or more model parameters.
3.2.3.1 Effect of window shape

Here we compare uniform and retinal input windows. Our initial model M1 (§3.2.1) has a uniform window. Changing to a retinal window improves performance in almost every condition. Recall that this amounts to removing most of the high-frequency units in the uniform window (fig. 3-1). As described in §3.1.6.1, these units are so
numerous that they can dominate the signature, especially when $C_2$ pools globally over all $(x, y, \lambda)$, and they are pooled so aggressively that the resulting signature loses distinctiveness. Notice that there is no improvement only for the (train small, test small) condition, where high-frequency information (relative to the object) isn’t available.

Training the retinal window model using multiple fixation points for each training object yields a further (albeit small) improvement. (Call this new model M2.) The likely reason is also discussed in §3.1.6.1: the fine bands of the retinal window will not always fall on the same part of the object. We may be testing with central fixation only, but intra-category variability means the same part may not always be at the object center. Training with multiple fixations yields a more representative set of training signatures. As expected, multiple training fixations has no effect on model M1 with its uniform window.
3.2.3.2 Global vs. no scale pooling

Here we test our hypothesis (from §3.1.5) that the signature should better preserve intra-object relationships in $\lambda$. We disable $C_2$ pooling over $\lambda$ altogether, yielding signatures as illustrated in fig.3-3. These signatures need to be aligned along $\lambda$ for comparison. For each test image, a range of alignments are attempted, reflecting ambiguity in the exact object size that would be present in a complex-scene scenario. We tried various degrees of uncertainty in the test object size. A reasonable uncertainty
of ±0.5 octaves yields a large boost in performance. This is new model M3.
3.2.3.3 Effect of window shape (no scale pooling)

We revisit window shape, this time in the absence of λ pooling. Contrary to what was seen in § 3.2.3.1, here the retinal window's performance advantage over the uniform window is insignificant. The uniform window's many high-frequency units are still over-aggressively pooled, but the results now remain segregated from the lower-frequency bands in the signature, no longer dominating them. Isolated within their own bands, the large difference in number of units and pooling aggressiveness is not affecting performance. We investigate this in the next experiment.
3.2.3.4 Effect of band masking

Here we investigate the contribution to performance of different bands by using attentional masking in \( \lambda \) (§3.1.2.3). Every model tested in fig. 3-8 includes the central band plus the bands indicated in the legend.

It turns out that omitting all the fine bands (fig. 3-2) has virtually no effect on performance. (The curve for this ‘all coarse bands’ model is almost completely hidden behind that of model M3.) Contrary to our conjecture in §3.1.6.1, the classifier seems to mostly ignore these bands – something which is now possible because they remain
segregated in the signature. This explains the equivalent performance of the two window shapes seen in the previous experiment: apart from the fine bands, these two models yield equivalent signatures.

As predicted in §3.1.6.2, all the coarse bands appear to be useful. Performance declines when bands more than 1 octave above the central scale are omitted. Note that in this isolated-object case, coarse bands do not contain contextual information, just lower-frequency information from the object itself.

Models which use only the fine bands, or only the single central band, perform poorly.
3.2.3.5 Effect of scale pooling range

Here we explore the middle ground between no λ pooling at all and global pooling. Results indicate that local λ pooling causes a slight decrease in performance. The bandwidth of the original Gabor filters appears to provide sufficient feature-level invariance over λ.

Figure 3-9: layout follows fig. 3-5.
3.2.3.6 Summary of optimizations

Here we simply recap the model improvements resulting from our experiments.

- **M1**: traditional HMAX model, with a uniform window and global pooling over λ (§3.2.1).
- **M2**: changed to retinal window (§3.2.3.1).
- **M3**: eliminated pooling over λ (§3.2.3.2).
3.3 Experiments: clutter

In this experiment we test robustness to some amount of clutter using two models, one having a uniform window and the other a retinal window. The models are M1 (§ 3.2.1) and M2 (§ 3.2.3.1) respectively.

The task is categorization of a fixated object as a distractor object is introduced and moved closer to the fixated object (see fig. 3-11, upper). This time, all objects are shown at size 1.0 oct, twice the foveola size (see fig. 3-4). The image set is as described in § 3.2.2. We train a single classifier using the uncluttered training set, then test it while distractors appear at various separation distances. Test objects appear at the fixation point with a distractor object the appropriate distance away on the right. The distractor object is randomly chosen from the test set, but must be of a different category than the fixated object. Scoring for each separation value is the average of the per-category scores.

Without any attentional masking, the retinal window is much more robust to clutter, as predicted in § 3.1.6.1. The retinal window introduces a bias in favor of the fixated object. Even in the zero-separation case, several of its scale bands remain completely uncluttered. The uniform window is dominated by high-frequency units, a significant number of which are affected even at relatively large separations.

3.4 Summary of contributions

As a biologically-inspired model of object recognition, HMAX supports several research goals: better understanding of the ventral visual pathway can yield deeper insight into the visual problem it solves, which can in turn help to improve artificial vision systems.

We investigated an HMAX model incorporating realistic retinal resolution. While a previous study [13] focused on recognition in the periphery, where performance is degraded, we believe the above goals are better served by focusing on the ventral stream's 'core competency': recognition of objects in central vision.
Figure 3-11: categorization performance for a fixated object when a distractor object appears at various distances, shown for two models, one having a uniform window and the other a retinal window. **Upper:** shows two clutter conditions: separation = 0 (distractor immediately adjacent) and separation = 0.4 (space between fixated object and distractor is 0.4 object diameters). Red dotted circles show the coverage of scale bands for the retinal window (every 1/2 octave shown). The innermost circle is the foveola. **Lower:** performance of the two models at various separation values.
A retinal capture window (§3.1.2.2) constrains what can be done in a single feed-forward pass, clarifying the boundary between HMAX and the larger process of scene understanding. This and the window shape itself led to a number of theoretical insights, including:

- A formal definition of the capture and input windows (§3.1.2) and the nature of the visual question HMAX answers for an object in a complex scene (§3.1.4).
- Limits on HMAX’s responsibility for invariance to position and scale of the entire object, and for tolerance to clutter (§3.1.4).
- A conjecture that preservation of information about the relative position of feature activations within the object itself is more important along $\lambda$ than in $(x, y)$ (§3.1.5) (confirmed).
- A conjecture that information from fine scale bands (fig. 3-2) that do not span the entire object could confer advantages (§3.1.6.1) in the isolated-object case (not observed), with clutter (confirmed), or with partial occlusion (not tested).
- A conjecture regarding use of contextual information (fig. 3-2) for recognizing small objects (§3.1.6.1) (not tested).
- Observations on the scale-invariant processing possible with a retinal input window and on useful computational properties of signatures it produces (§3.1.7) (not tested).

The first set of experiments (§3.2) tested categorization of isolated objects and generalization from a training size to various test sizes. Standard HMAX [44, 29] (fig. 3-10, model M1) is less tolerant of scaling than commonly assumed, with performance dropping by 50% at a size change of around 1.3 octaves.

The model change which most improved performance was the elimination of global pooling over $\lambda$ from the final pooling stage before the signature (fig. 3-10, model M3), with the goal of better preserving relative scale relationships between feature activations as proposed in §3.1.5. This creates signatures that retain separate scale bands as diagrammed in fig. 3-3. As noted in fig. 3-3, the ventral stream might achieve the same goal more easily via additional $(S, C)$ layers. But under our method, best
results were obtained without even local scale pooling. Scale invariance provided by layer $S_1$’s Gabor filter bandwidth (§3.2.1) seems to be sufficient for this task. The performance gain over M1 is around +10% even when no size change occurs; under size change performance is vastly superior.

An unexpected finding is that in the isolated-object scenario, spatial frequencies not available across the entire object (fine bands) do not significantly affect performance. Experiments in which these bands were masked out (§3.2.3.4) or in which they remained segregated in separate bands in the signature where the classifier could learn to ignore them (§3.2.3.3), yielded nearly identical results. With fine bands eliminated, the final input window used to compute the signature is the same regardless of capture window shape (uniform or retinal) before masking. A retinal-shaped input window was found to help in the clutter scenario (§3.3), where having fine bands which remain uncluttered is an advantage.

**Acknowledgements**

T. Poggio sparked my interest in modeling realistic retinal resolution and led our efforts to interpret it in terms of i-theory [36]. Leyla Isik and Gadi Geiger participated in discussions. L. Isik and Joel Leibo previously investigated crowding in the periphery using a similar model which predated i-theory [13]. J. Mutch accepts responsibility for the change in emphasis of the present project from peripheral to central vision. T. Poggio provided valuable advice throughout. Gemma Roig and Daniel Harari offered insights and encouragement on several occasions.
Chapter 4

Discussion

Summary discussions of completed work appeared at the end of preceding chapters (§2.5 and §3.4). The focus of this concluding chapter is on future directions. We consider both general approach and specific issues.

4.1 General approach

4.1.1 Feedforward core

This work, in particular §3, assumes the current consensus view [5] that for recognition (not learning), the ventral stream's design is fundamentally feedforward, with tasks more complex than immediate recognition involving top-down modulation of this feedforward core by higher-level processes. The most obvious such modulation is the selection and placement of input windows (§3.1.2) that takes place during scene understanding.

HMAX and CNNs both model just this feedforward core. This fact is commonly misrepresented as a claim that 'vision is entirely feedforward'; in reality it simply reflects the state of research. Feedforward models are much simpler theoretically than recurrent ones, and data obtained from human or animal experiments is much easier to interpret for experiments in the immediate recognition regime.

For biological brains, there is obvious evolutionary pressure favoring a design
in which feedforward recognition handles as much of the overall problem as possible. Every synaptic jump between the retina and the initiation of a physical response adds a delay measured in milliseconds, which is significant in the context of the timescale of body movement. Unnecessary looping is contraindicated.

In § 3.1.7 we touched on the idea that signatures resulting from an input window having retinal shape naturally encode has-a relationships between objects and parts. This adds plausibility to the concept of a feedforward core providing input to various higher-level processes. Such signatures are a richer set of building blocks in which information concerning the hierarchical structure of the scene has not been ‘flattened out’.

The design of the primate retina and ventral stream is such that delivery of this ‘nested’ representation to higher areas is essentially the default; masking is needed to prevent it. It’s usually a good bet that the brain will take advantage of readily available, useful information.

4.1.2 Biological fidelity

Fig. 4-1 illustrates two approaches to the study of object recognition in terms of Marr’s three levels of analysis (see legend).

- Ventral stream modeling (VSM): starting with experimental data, try to infer the algorithm which the ventral stream implements. Insights from the computational level (on the nature of the object recognition problem itself) can help with the interpretation of experimental data and with writing a computer implementation, providing a basis for educated guessing when critical aspects of the ventral stream algorithm are uncertain. Use simulation results to evaluate choices and inform new experiments. HMAX is an attempt at an algorithmic-level description of the ventral stream (in feedforward mode).

- Pure computer vision (PCV): ignore the brain, to the degree this is possible. (Our intuition about vision comes from having a biological vision system.) Assess the problem at the computational level, design and implement algorithms
Figure 4-1: Marr's levels of analysis [26], illustrated for two approaches to the study of object recognition, ventral stream modeling and pure computer vision (see text). In general terms, the three levels are:

- **Computational**: the nature of the problem and the challenges it poses.
- **Algorithmic**: a method for solving the problem, in terms of specific representations and processes.
- **Implementation**: how is the algorithm physically realized?

Red bidirectional arrows represent the fact that better understanding of any one level can lead to better understanding of the others.
to solve it, and use simulation results to assess and refine.

If one’s priority is to understand the brain, VSM is the obvious approach to take, although PCV plays an important supporting role. Lacking the brain as a guide, PCV relies entirely on assessing the problem itself, and better understanding of object recognition at the computational level benefits VSM.

For those interested only in machine vision, the choice of approach is less obvious. VSM constrains the space of potential algorithms to those which could potentially be the ventral stream’s. Whether one considers this harmful or helpful depends on one’s tendency to optimism or pessimism. These are algorithms suitable for implementation with neurons, and which evolution could conceivably find; neither restriction is relevant for computers. Better algorithms likely exist outside this space. On the other hand, the space of all possible algorithms is much larger. How cleverly can we search that space? How common are good solutions? Early PCV researchers were definitely optimists [33].

Over the last few years, the ascendence of CNNs (partly attributable to increased computing power and availability of labeled data) arguably has VSM ‘leading’ over PCV. Backpropagation comes from multilayer perceptrons, which were themselves neurally inspired, and the feedforward convolutional architecture of CNNs was directly inspired1 by that of the ventral stream. These comprise the defining features of the CNN class; however, in the case of convolution, the ventral stream concept has been only partially adopted. CNNs are convolutional only in \((x, y)\). But bandpass filtering by V1 simple cells effectively creates a scale \((\lambda)\) dimension.2 In §3 we argue for the importance of \(\lambda\) as a third convolutional dimension, and we explore the deep relationship between an explicit \(\lambda\) dimension and retinal resolution.

Over time, the benefits to machine vision of remaining within the VSM paradigm will diminish. But at the moment, with one of the key biologically-inspired features of CNNs – the current performance leader – only partially explored, VSM seems

1We consider ‘biologically inspired’ to mean an idea which originated from study of the brain. Theoretical explanations which may emerge later do not affect this status.

2While CNNs tend to learn V1-like features in the first layer, they do not systematically cover \((\lambda, \theta)\) and \(\lambda\) does not persist as a separate convolutional dimension in subsequent layers.
promising.

4.1.2.1 What should models learn?

Within the VSM paradigm, a common question is: should our full computer implementation try to learn $S_1$ features, or should we impose them? Learning them is a good way to test feature-learning—it should yield Gabor filters for $S_1$ (as in §2)–but assuming any such test has been passed, what is gained by learning them for every model we train?

V1 is at least learned (or tuned) after birth. There is, however, a ‘pure learning’ school of thought which abhors fixing even parameters whose values can be reliably inferred from the ventral stream or via computational-level considerations. At its extreme this fits neither of the paradigms in fig. 4-1; domain-specific analysis ceases and we rely entirely on a powerful learning algorithm. This is another case of optimism vs. pessimism, this time with respect to the learning algorithm. Is it as flexible as evolution? How much data and time do we have?

Based on practical experience with limited resources, our approach is usually to avoid learning more parameters than we have to.

4.1.3 Next steps

Of the two main research paths described in this thesis, we believe further exploration of the retinal resolution path (§3) more likely to be productive in the short term. It is a less-explored topic than bottom-up feature learning in convolutional architectures (§2), and our current progress has already been shown to improve object recognition performance.

By contrast, our attempts at improving object recognition performance using the feature learning algorithm of §2.2.2 have so far not succeeded. The top-down feature learning used in CNNs outperforms all bottom-up approaches. Its reliance on large quantities of labeled training data means it is probably not the ventral stream’s feature learning algorithm, but so far it has proven tough to beat.
For upcoming work on the retinal resolution path, our first step will be to adopt CNN-style backpropagation feature learning for $S_2$ and above. This will enable fair performance assessments relative to the current state-of-the-art. (As the resulting model will possess the key characteristics of both HMAX and CNNs, it's unclear what it should be named.)

One reason backpropagation and scale bands have not previously been combined is difficulty of implementation. This thesis improves the situation in two ways:

- With a retinal input window, every scale band has the same number of units, so model layers can be implemented as 4D rectangles (tensors) in $(x, y, \lambda, f)$.

- In the general case (any shape input window), models can be implemented using CNS (§ A). CNS has support for convolution-like operations across bands of dissimilar size and sampling that are not present in other platforms commonly used to implement CNNs.

4.2 Specific issues

Here we briefly sketch some specific ideas for further research. Many of these are related to retinal resolution (§ 3); see also § 3.1 which contains several propositions which remain untested.

4.2.1 Deeper hierarchy

In the best performing HMAX models, the top layer is $C_2$. As each $(S, C)$ layer pair corresponds to a cortical area, this is not as deep as the ventral stream (V1, V2, V4, IT). Adding more $(S, C)$ layers has always lowered performance, likely due to feature learning; switching to CNN-style feature learning may help in surmounting this obstacle.
4.2.1.1 No shifting

In §3.2.3.2 we significantly improved performance by disabling global pooling over \( \lambda \) in top layer \( C_2 \), which was destroying relative scale information among feature activations. The resulting signatures retain topology in \( \lambda \) and must therefore be aligned prior to classification or comparison (fig.3-3). This is easy in a digital computer, but the ventral stream has no obvious mechanism for large-scale shifts of variable magnitude.

A deeper hierarchy could allow more gradual pooling over \( \lambda \), eliminating the \( \lambda \) dimension by the top layer without discarding distinctive information (§4.2.1.2).

4.2.1.2 BOF degree

The computer vision concept of a ‘bag of features’ (BOF) refers to a representation which is sensitive to the presence of visual features but insensitive to their relative position. This is easily achieved for each feature by pooling over position. The concept is easily extended to include scale.

In using a BOF, a designer hopes to increasing invariance to transformations without losing distinctiveness. But in achieving this, not all BOFs are equivalent. For example, if the features being pooled are very simple (or, at the extreme, just one pixel across), pooling is more likely to destroy discriminability.

BOF is typically not used to describe hierarchical models, but the output of every column of \( C \) cells (within the same layer and sharing the same receptive field) can be seen as a local BOF. Possibly a major reason for the hierarchy itself is to avoid losing discriminability by pooling a little bit, then computing higher-order features before pooling again. Finding a good combination of feature complexities, pooling ranges, and number of hierarchical levels is still more art than science.

A better theoretical understanding of what we loosely refer to as BOF degree might be helpful: the relationship (possibly ratio) between feature complexity and pooling range.
Figure 4-2: Two possible shapes for receptive fields that span multiple scales. (A) is analogous to a uniform window (§3.1.2.1), sampling more from finer bands than coarser ones. (B) is analogous to a retinal window (§3.1.2.2), sampling each band equally.

4.2.1.3 Multiscale receptive fields

One reason pooling over $\lambda$ is currently losing distinctiveness is that our existing $S_2$ features are very simple along the $\lambda$ dimension – just one band wide. This is somewhat analogous to a feature one pixel wide in $(x, y)$. The subsequent pooling step thus has a high BOF degree (§4.2.1.2). Introducing features that are extended in $\lambda$ raises a question about the receptive field shape of $S$ cells; see fig.4-2. If any pooling over $\lambda$ is going to occur before pooling in $(x, y)$ has become global, the same question arises for $C$ cell RFs.

We tend to prefer shape B (retinal-like), at this point mainly on aesthetics. They are also easier to implement for the reasons given in §4.1.3.

Note that if features and/or RFs were learned from motion, shape B would arise from centered scaling, which is easily produced by egomotion.

4.2.2 Small or unfixated objects

When a fixated object is smaller than the foveola, or we are attending to an unfixated object, no scale band of a retinal capture window can see $N$ cycles (§3.1.2.2) on the object. Consider the finest band that does span this object, extracting $N_{obj} < N$
cycles. The corresponding band for a centrally-fixated object larger than the foveola would be one of the coarse bands as shown in fig. 3-2.

4.2.2.1 Pass-through concept

A possible concern with small objects is that they might not activate $S$ cells near the top of the hierarchy very well, where receptive fields are large. One solution to this would be to include $C$ cells below the top layer in the final signature as well. They would first have to undergo further pooling to match the invariance achieved in the top layer.

This idea generalizes to a concept which could be applied at all levels and possibly improve more than just the small-object scenario. Presently each $S_i$ layer projects its input, which is encoded using feature set $F_{i-1}$, onto a completely different feature set $F_i$. Our intuition here is that this is like keeping only $i^{th}$ order correlations and discarding lower-order correlations.

Rather than throwing away lower-order feature activations, they could instead be pooled and downsampled. Suppose $|F_{i-1}| = 300$. Feature set $F_i$ could consist of 500 new $i^{th}$ order features plus pooled versions of $F_{i-1}$, giving $|F_i| = 800$. In layers $(S_i, C_i)$, $f$ indices 1-300 would refer to pooled versions of lower-order features, and $f$ indices 301-800 to new features.

Under this scheme, features of mixed order are available, not just to the final classifier, but to $S$ cells anywhere in the hierarchy. Features originating at any level could potentially 'pass through' all the way to the top, being repeatedly pooled and downsampled, or we could easily impose a limit.

4.2.3 Input window vs. object location

In a computer vision sliding-window setup, the (uniform) input window is implicitly the putative object boundary. In §3.1.4 we decoupled these concepts slightly. One might consider decoupling them completely. During training, part of the information stored with a signature could be the location of the fixation point relative to
the object. At recognition time, if that training signature is the best match, that information could help plan the next fixation.

A single training signature could actually have multiple associated labels. Fixating someone's nose, the labels 'nose' and 'face' are equally valid – just at different scales. This is a second conceptual decoupling: a signature does not represent 'an object'.

4.2.4 Top-level classifier

Fine scales did not contribute to performance in the isolated-object experiments (§3.2). A possible explanation could be the linear final classifier used. When learning a category, the information in finer bands is more volatile because the fixation point is not always on the same part of an object. An overly-simple classifier might not be able to use this information effectively and just end up using the coarser bands.

Switching to CNN-style learning might give us a better classifier 'for free', as the fully-connected (non-convolutional) layers at the top are a nonlinear classifier.

4.2.5 Context-assisted recognition

In §3.1.6.2 we discussed the possibility that context information (from outside the object) could aid recognition for some objects, especially when the object is small. This would be more often true for objects that are parts of other objects.

An interesting experiment might be to include context for all objects in training, and have the classifier learn for which categories context is informative.

4.2.6 Connection with minimal images

There are likely to be deep ties between the retinal resolution work (see in particular §3.1.4) and that of Ullman et al. on minimally recognizable images [50].
Appendix A

CNS: a GPU-based framework for simulating cortically-organized networks

Computational models whose organization is inspired by the cortex are increasing in both number and popularity. Current instances of such models include convolutional networks, HMAX, Hierarchical Temporal Memory, and deep belief networks. These models present two practical challenges. First, they are computationally intensive. Second, while the operations performed by individual cells, or units, are typically simple, the code needed to keep track of network connectivity can quickly become complicated, leading to programs that are difficult to write and to modify. Massively parallel commodity computing hardware has recently become available in the form of general-purpose GPUs. This helps address the first problem but exacerbates the second. GPU programming adds an extra layer of difficulty, further discouraging exploration.

To address these concerns, we have created a programming framework called CNS ('Cortical Network Simulator'). CNS models are automatically compiled and run on a GPU, typically 80-100x faster than on a single CPU, without the user having to learn any GPU programming. A novel scheme for the parametric specification of network connectivity allows the user to focus on writing just the code executed by
a single cell. We hope that the ability to rapidly define and run cortically-inspired models will facilitate research in the cortical modeling community. CNS is available\(^1\) under the GNU General Public License.

### A.1 Introduction

#### A.1.1 Definitions

For the purpose of this report, we define a 'cortical' model to be a network model which consists of some number of N-dimensional 'layers' of cells, where each layer encodes some N-D feature space. It is common for at least some of the dimensions to be topographically mapped, meaning that physical proximity in the layer corresponds to proximity in the feature space. N, and the feature space, can be the same or different from layer to layer. All the cells in a layer must be of the same type, i.e., each maintains its own values of the same set of variables, using the same algorithm. Connectivity between cells may be completely arbitrary, but often consists of a repeating pattern.

This class of models includes, but is by no means limited to:

- Convolutional networks [23, 15].
- HMAX [45, 29].
- Hierarchical Temporal Memory [8].
- Deep belief networks [9].
- Detailed spiking models of cortex.

Some of these are 'static' models, requiring only a single pass through the network for a given input. Others have dynamics that require iteration over many time steps.

#### A.1.2 Motivation

Cortical models commonly contain a large number of units and are therefore computationally expensive. However, they are highly amenable to parallelization. The

layered structure of cortical models maps very well to the architecture of modern GPUs, which are optimized to perform the same operation at every point in an array of data. GPUs have evolved from graphics accelerator cards, where the array elements are pixels. Over the last few years the APIs for these cards have been opened up and generalized to encourage the acceleration of non-graphical algorithms that can benefit from the same architecture. Current GPUs have hundreds of parallel processors and can typically run suitable algorithms 80-100x faster than a single CPU. However, these performance gains are not free. A GPU's processors all still share a common memory, which becomes the bottleneck of the system. To achieve optimal performance, GPU programmers must code their algorithms so that the processors access memory in coordinated patterns. This adds another layer of difficulty to models that can already be somewhat challenging to program.

Individual cells in cortical models typically perform fairly simple functions. Difficulties in programming these models usually arise in keeping track of the connectivity between cells. There are two common approaches:

- **Enumerate every synapse.** With this approach, one can represent any network architecture, but for networks having regular patterns of connectivity, it is extremely wasteful. Memory usage increases drastically, and because memory access is the bottleneck in multiprocessing systems, so does processing time.

- **Matrix operations.** This is a common abstraction that works for some simple hierarchical models. Layer $n$ is generated from layer $n - 1$ via some matrix operation such as N-D convolution. This is very space-efficient and easy to implement in a language such as MATLAB. It is also fairly straightforward to implement on a GPU, and to program arbitrary response functions in place of the dot product implied by convolution. However, in more complex situations this abstraction becomes limiting. Actual cortical areas typically receive convergent input from multiple pathways that have been processed differently. Operations may have been carried out at different resolutions, or the number of processing steps may have been different. Thus, the indices of cells in two
different input layers no longer have the same meaning, and it is difficult to
define a matrix operation that combines them. Most work in cortical modeling
has simply focused on the subset of models for which these difficulties do not
arise.

Of the two barriers to exploration of the cortical model space – programming time
and run time – GPU programming alone can only address the latter, and at the
expense of the former. A framework is needed that can address both issues.

### A.1.3 Overview of CNS

CNS is a rapid development environment for cortical models. Models are compiled
and run automatically on NVIDIA GPUs, often running 80-100x faster than on a
single CPU, without the user having to learn any GPU programming.

Most aspects of a model are defined via MATLAB scripts, including:

- The number of layers.
- The dimension, size, and cell type of each layer.
- The variables associated with each cell type, and their initial values.
- The connectivity between cells.

The process of running models, loading input data, and pulling back results is also
controlled via MATLAB. Variables are referred to by name – the user does not need
to be concerned with where they are stored in GPU memory.

There are two options for specifying connectivity. As in many other simulators,
synapses may be explicitly enumerated: a given cell can list any number of presynaptic
cells, which can be in any layer. For models having regular patterns of connectivity,
CNS uses a scheme in which each cell explicitly retains its grid position (the center
of its receptive field) in a real-valued feature space which is meaningful across layers.

For example, in a vision model, the dimensions of this common feature space would
probably include retinal position, and even cells several steps removed from the input
would still know their center coordinates in retinal space. Under this scheme, cells
can infer their inputs based on proximity in the common feature space. By appropri-
ate relative arrangement of the coordinate grids for each layer, any of the standard connectivity patterns (e.g., valid and full convolution, sub- and super-sampling, etc.) can be achieved, as well as many others. Once defined in this way, connectivity is handled by the framework. An individual cell can make requests of the framework, for example, it can request the indices of its $n$ nearest neighbors in layer $z$. This division of labor allows programmers to focus mainly on the code being executed by a single cell.

A 'kernel' is the code each cell executes during an iteration of the network. Cells of the same type share the same kernel. Kernels run on the GPU and are the only parts of a CNS model that must be written in C/C++. Even when writing a kernel, however, the programmer remains isolated from the complexities of the GPU. A kernel is written from the point of view of a single cell, so the programmer is not responsible for any thread scheduling. Nor is the programmer required to know where variables are stored in GPU memory. CNS provides named macros that the kernel can call to, among other things:

- Read and write the current cell’s variables.
- Find input cells via proximity in the feature space, as described above.
- Read the values of other cells’ variables.

During model development, CNS can be thought of as a compiler, in that programmers do not need to read or modify any of CNS’s code. By way of comparison, the 3-D convolutional network package with backpropagation described in §A.2.3, written using CNS, comprises about 300 lines, while CNS itself is about 10,000 lines as of this writing.

CNS is licensed under the GNU General Public License. The software and programmer’s manual are available at http://cbcl.mit.edu/jmutch/cns.

### A.1.4 Structure of this report

The remaining sections of this report are as follows:

- §A.2 introduces three example packages written in CNS.
• § A.3 discusses some CNS concepts in more detail, using the example packages for illustration.
• § A.4 describes the process of developing and running a CNS model.
• § A.5 provides some internal details on how CNS maps models to the GPU architecture.
• § A.6 lists some limitations of CNS.
• § A.7 summarizes CNS’s contributions to the cortical modeling community.
• § A.8 discusses some possible future work on CNS.

This is intended as a high-level introduction to CNS. Exact syntax is not covered beyond a few examples. For a complete description of all CNS syntax and options, see the programmer’s manual [28].

A.2 Example packages

In CNS, a 'package' is a collection of cell types that is used to construct models. For example, the 'HH' package described below implements several types of Hodgkin-Huxley cells. Once you have a package, you can then build networks consisting of those types of cells. The same package can be used to define many specific models, having different numbers of layers, of different sizes, with different connectivity, etc.

In this section we briefly describe three packages that we have developed with CNS. Subsequent sections will refer back to these examples to illustrate various CNS concepts.

A.2.1 'HH' package: Hodgkin-Huxley spiking models

The Hodgkin-Huxley (HH) package is used to build models made up of 2-D layers of spiking neurons, similar to the cortical laminae. Fig.A-1 shows part of three layers of such a model:

• IN: the input layer. Each cell has its own list of preprogrammed spike times.
• F1: a layer of fast-spiking inhibitory cells, each receiving input from a local
Figure A-1: Schematic of part of a model made up of Hodgkin-Huxley spiking neurons (see §A.2.1), implementing monosynaptic excitation with disynaptic inhibition. IN: input layer. F1: fast-spiking inhibitory cells (Wang-Buzsaki model). P1: pyramidal cells (Golomb-Amitai model). IN-P1 connections, and other layers, are not shown. The full model contained 10,000 cells.

- P1: a layer of pyramidal cells, each receiving input from a local region of IN and F1 (latter connections not shown).

In these models, cells in a layer are assigned to grid positions in a 2-D feature space, but with some gaussian noise added. Connectivity is also noisy; for example, each F1 cell receives input from a local region of IN cells, but with the probability of a connection decreasing with distance from the center of the region. Thus, connectivity in these models is defined using the explicit synapses method: each cell in a model lists its presynaptic cells.

Models built with this package are dynamic. Each cell maintains its own set of Hodgkin-Huxley state variables. Each iteration of the network represents a small time step, during which each cell polls its presynaptic cells to see which ones are spiking and updates its state variables according to a set of differential equations. CNS can track the values of selected state variables as they change; fig.A-2 shows a voltage trace for one cell.
For a model with 10,000 neurons and 330,000 synapses, CNS was able to process 5,000 time steps per second on a GTX 285 GPU.

A.2.2 ‘FH’ package: HMAX-like feature hierarchies

The FH package allows you to build models of the HMAX [45, 29] class. HMAX models the initial feedforward stage of object recognition in the ventral visual pathway. It extend the idea of simple and complex cells [11] to form a hierarchy in which alternating template matching and max pooling operations progressively build up both feature selectivity and invariance to position and scale. Fig. A-3 illustrates one such model [29] as it is expressed in CNS.

HMAX is scale invariant: each stage of processing is carried out at multiple scales. Thus, each stage of the model is actually represented by many CNS layers, one for each scale. For the model in fig. A-3, the stages are:

- S1: Many different resolutions of the original image, produced by bicubic interpolation.
- S1: Applies gabor filters of several orientations at every position and scale. Each scale is now of size $F_1 \times Y \times X$, where $F_1$ is the number of orientations.
- C1: Independently for each orientation, computes the maximum response over
Figure A-3: An example of an HMAX model (see §A.2.2). Each step is performed at multiple scales, only three of which are shown here.

- S2: Computes the response to many stored templates at every position and scale. Now in each scale we have $F_2 \times Y \times X$ units, where $F_2$ is the number of templates.

- C2: For each template, finds the maximum response over all positions and scales. This results in a feature vector which can be fed into a classifier.

These models are computed in a single bottom-up pass for a given input image. The gabor filters in S1 and the stored templates in S2 are shared by all the cells in those layers (and are not shown in fig. A-3).

The FH package replaces an older CPU-based library called FHLib [27] which was
used for the experiments in [29]. For one large model, CNS outperformed FHLib by a factor of 97x on a GTX 285 GPU.

A.2.3 'CN' package: convolutional networks for 3-D image segmentation

The 'CN' package is a CNS reimplementation of the code used in [15]. It is used to train and run convolutional networks that perform segmentation in 3-D electron microscope images of brain tissue. These networks are similar to those of Lecun [23] except that the filters have three spatial dimensions and there is no subsampling. The model architecture is illustrated in fig. A-4.
Each $X_i$ layer is four-dimensional, containing the value of $F_i$ different features at each position in a 3-D cube. (For the input image $X_0$ there is only one feature: the pixel value.) The features are different in each stage. Each $W_i$ can be viewed as $F_i$ four-dimensional filters which, via convolution over $X_{i-1}$ (and then adding the appropriate bias from $B_i$), produce layer $X_i$. Note that convolution only occurs over the three spatial dimensions; for the feature dimension, $X_{i-1}$ and the filter are the same size.

The goal of the training phase is to learn all the $W_i$ and $B_i$ layers. This is done via the backpropagation algorithm in three passes (mathematics in [15]):

- **Forward pass:** starting with the input in $X_0$, compute the outputs of each $X_i$ layer from bottom to top.

- **Backward pass:** starting with the desired output in $X_n$, compute the error term ('sensitivity') for each $X_i$ from top to bottom.

- **Weight update:** update each $W_i$ and $B_i$ layer.

Note that unlike the filters and features in FH models (§A.2.2), here the weights and biases need to change during a network iteration. Thus, we treat them as layers, and they have their own kernels which perform the update at the appropriate time.

Once the network has been trained, only the forward pass is needed to perform segmentation.

The CNS implementation of this class of models ran about 100x faster than the previous code, based on timing using a single CPU. Ongoing development of these models continues under CNS.

### A.3 CNS concepts

In this section we run through the key concepts of CNS, using the packages in §A.2 as examples.
A.3.1 Layers and groups of layers

The basic architectural unit of a CNS model is the layer, which is an N-dimensional array of cells that are all of the same type (see § A.3.3). Practically speaking, a model can have up to several hundred layers. Note that layers do not have to be arranged in any kind of hierarchy or directed acyclic graph (DAG).

Multiple layers of the same type can be designated as a group. The main purpose of this is to allow them to share some common data. For example, in the FH package, all the S1 layers (of different scales) are designated as a group so they can share the same set of precomputed gabor filters.

A.3.2 Execution order

By default, a single network iteration consists of computing all cells in all layers exactly once, in parallel. HH models work like this. Internally, of course, everything isn’t computed exactly at once. A double-buffering system is employed to guarantee that during iteration \( t \), all the values a kernel reads from other cells come from iteration \( t - 1 \).

For some models, it wouldn’t make sense to compute all the layers at the same time. For example, cells in FH models do not have states that evolve over time. There is no point in computing layer \( n \) until layer \( n - 1 \) has been computed. In this case, a network iteration is broken into steps, with layers assigned different step numbers. In fig. A-3, for example, all the SI layers are assigned step 1, the S1 layers step 2, the C1 layers step 3, etc.

CN models are similar to FH, except that a single iteration consists of a forward pass, a backward pass, and a weight update. The \( X_i \) layers get computed at two different points in one iteration, i.e., they each have two different step numbers, and perform a different computation in each.

Note that there is no unit of execution smaller than a single layer.
A.3.3 Cell types

A cell type defines:

- The dimensionality \((N)\) a layer of cells of that type must have. For example, all the layers in HH models are 2-D, but in CN models the \(X_i\) layers are 4-D, the \(W_i\) layers are 5-D, and the \(B_i\) layers are 1-D.

- The constants and variables \((\textit{fields})\) associated with each cell. (Fields can also have layer scope, synapse scope, etc.; see §A.3.4.)

- The kernel used to update a cell during a network iteration (see §A.3.5).

Just like a class in object-oriented programming, a cell type can be a subtype of a parent type. All the cell types in a package form a type hierarchy (not to be confused with a model having hierarchical structure). Every package must have a 'base' type which is the root of the type hierarchy. A cell type inherits the following properties from its parent type:

- The dimensionality of a layer. Note that this cannot be overridden.
- All fields.
- The kernel. This can be overridden.

Cell types can be \textit{abstract}, which means they exist only to declare a common set of properties that are then inherited by subtypes. In many packages, the 'base' type will be abstract. You cannot create layers of cells of an abstract type; you must use a non-abstract subtype.

Table A.1 shows the type hierarchy for all three example packages.

A.3.4 Fields and scope

In CNS, a named numeric quantity associated with a cell is called a \textit{field}. For example, in HH models, membrane voltage \((V_{\text{m}})\) is a field, and each cell maintains its own value of the membrane voltage. For cells, fields are analogous to the data members of a class or structure in object-oriented programming. Each cell type inherits its parent type's fields and may also define its own fields. The definition of a field includes:
<table>
<thead>
<tr>
<th>Type</th>
<th>Fields</th>
<th>Layer or group</th>
<th>Cell</th>
<th>Synapse</th>
<th>Kernel</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>HH package</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>base*</td>
<td>V_m t_last</td>
<td>no</td>
<td>Y x X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>input (spike times)</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>passive*</td>
<td>C_m g_L E_L</td>
<td>g, r</td>
<td>template</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dendrite</td>
<td>parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>active*</td>
<td>E_Na E_K</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ga</td>
<td>h, n, b, z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wb</td>
<td>parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| FH package | | | | | | |
| base* | val | no | F x Y x X |
| raw-im | no |
| scaled-im | yes |
| s-fixed* (filters) | template |
| ndp | parts |
| e* (rf size) | template |
| max | parts |
| avg | parts |
| s-learned* (features) | template |
| grbf | sigma |
| grbfnorm | sigma |

| CN package | | | | | | |
| base* | val | no | undefined |
| layer* | no | F x Y x X x D |
| input | |
| computed* | sens | template |
| hidden | parts |
| output | correct | parts |
| weight | eta | val | yes | F_in x Y x X x D x F_out |
| bias | eta | val | yes | F_out |

**Table A.1:** Cell types for the example packages. Indentation in the 'type' column denotes inheritance, and * denotes an abstract type. Italicized fields are read-write, i.e., variable. Note that a cell type inherits all its parent type's fields.
• Field name. Used to identify the field, both in the MATLAB interface and inside kernels.

• Data type: either single-precision floating point or 32-bit signed integer.

• Scalar or vector.

• Read-only or read-write. Read-only fields (constants) cannot change their values during a model iteration on the GPU, but they can be changed between iterations from within MATLAB. Read-write fields (variables) may be changed during a model iteration, but only for the current cell, i.e., each cell is responsible for updating its own variables.

• Default value: the value a field will have when a model is initialized. May be overridden at model initialization time. If no default value is defined, then one must be given at model initialization time.

• Scope: see below.

One key way in which CNS fields are not like data members in object-oriented programming is that you can also define their scope. A field can have:

• Cell scope. Each cell maintains its own value of the field. (If the field is a vector, each cell maintains its own vector value. This applies to all the scopes listed here.)

• Synapse scope. Available when using explicit-synapses connectivity (§A.3.6). A separate value of the field is maintained for each synapse. This can obviously take a lot of memory.

• Layer scope. Each layer will have one value of the field that applies to the whole layer. Such fields are always read-only.

• Group scope. One (read-only) value of the field is maintained for an entire group of layers (see §A.3.1).

• Model scope. One (read-only) value of the field is maintained for an entire
network model.

Table A.1 shows many of the fields defined by the example packages.

One more kind of field is also available. In some models it is desirable for many cells to have access to the same large multidimensional array of static data. One way of doing this would be to use the same approach the CN package uses to share $W_i$ and $B_i$ values, which is to make such arrays into layers of their own. In CN models, $W_i$ and $B_i$ are not static, so this is the only option. In the FH model shown in fig. A-3, however, the S1 stage’s gabor filters and the S2 stage’s stored features are static during a network iteration, so it would be nice to not have to complicate the network structure by making them separate layers. CNS allows static arrays like this to be defined as fields having layer, group, or model scope.

### A.3.5 Kernels

A *kernel* is a function that updates a cell’s variables during a network iteration. As with methods in object-oriented programming, a kernel is written from the point of view of a single cell. Unlike object-oriented programming, however, each cell type has exactly one kernel.

Kernels are written using a limited subset of C++ (mostly just C), supplemented by macros generated by CNS. You may provide an arbitrary block of C/C++ code, subject to these conditions:

- Only statements that are permissible in function scope (inside a function body) are allowed.

- The only standard library functions available are those in the 'math' library.

- No dynamic memory allocation is permitted.

Access to fields and other properties you have defined for the various cell types is done via macros that CNS provides. This spares you from having to worry about how data is laid out in GPU memory, etc. Fig. A-5 shows part of a kernel from the CN package; all the red symbols are macros generated by CNS, based on the definitions for cell
types in that package.

When you are writing a kernel, you can ask CNS to list all the macros available to you. CNS provides macros to:

- Read and write fields. (For example, the READ_* macros, WRITE_VAL, ZP, ZW, and ZB in fig. A-5.)
- Retrieve current cell coordinates. (For example, THIS_F in fig. A-5.)
- Find receptive fields. (For example, the *_RF_NEAR macros in fig. A-5.)
- Retrieve layer dimension sizes. (For example, the *_SIZE macros in fig. A-5.)
- Loop through explicit synapses, if any.
- Retrieve the current iteration number and other miscellanea.

Closely related kernels are often very similar. For example, one might want to define a cell type having a kernel like the one in fig. A-5, but with the dot product operation, or perhaps the sigmoid nonlinearity, replaced by something else. To avoid having many kernels that are slightly-modified copies of one another, CNS allows you to write a template kernel for an abstract parent type. The template kernel will contain most of the logic, leaving 'blanks' for the specific operations. Then the various subtype kernels provide different code snippets (parts) that 'fill in' those blanks for each subtype. This feature is used in all three example packages; see the 'kernel' column in table A.1.

### A.3.6 Connectivity

As discussed in §A.1.2 and §A.1.3, CNS has two ways to specify cell-to-cell connectivity:

- HH models use the explicit synapses method: each cell lists each of its presynaptic cells. This is necessary because cells in HH models do not occupy regular grid positions in a common feature space.
- FH and CN models use the common coordinates method. Here we expand a bit on that topic.
// Retrieve the size of our 4-D filter.
int fSize = WEIGHT_F_SIZE(ZW);
int ySize = WEIGHT_Y_SIZE(ZW);
int xSize = WEIGHT_X_SIZE(ZW);
int dSize = WEIGHT_D_SIZE(ZW);

// Find our corresponding RF in the previous layer.
int y1, x1, d1, dummy;
GET_LAYER_Y_RF_NEAR(ZP, ySize, y1, dummy);
GET_LAYER_X_RF_NEAR(ZP, xSize, x1, dummy);
GET_LAYER_D_RF_NEAR(ZP, dSize, d1, dummy);

// Compute response to the 4-D filter.
float v = 0.0f;
for (int f = 0; f < fSize; f++) {
    for (int k = dSize - 1, d = d1; k >= 0; k-, d++) {
        for (int j = xSize - 1, x = x1; j >= 0; j-, x++) {
            for (int i = ySize - 1, y = y1; i >= 0; i-, y++) {
                float p = READ_LAYERVAL(ZP, f, y, x, d);
                float w = READ_WEIGHTVAL(ZW, f, i, j, k, THIS_F);
                v += p * w;
            }
        }
    }
}

v += READ_BIASVAL(ZB, THIS_F);

// Apply sigmoid nonlinearity.
v = 1.0f / (1.0f + expf(-v));

// Store result.
WRITE_VAL(v);

Figure A-5: Part of a kernel from the CN package. This code performs the forward pass. It computes the value of a single cell in an X_i layer by overlaying the appropriate 4-D weight array from W_i onto the cell's 4-D receptive field in X_{i-1}, computing the dot product, adding the appropriate bias from B_i, and applying a sigmoid nonlinearity. Symbols in RED are macros generated by CNS; everything else is standard C/C++. See § A.3.5 for details.
Figure A-6: Common coordinate positions for several layers of an FH model. The C1 cell shown is a corner cell of a layer that has been arranged to perform a 10x10 valid convolution over the layer representing the first S1 scale. The second S1 scale’s cells do not line up with either of the other two layers shown; however, it is still possible to define a pooling operation over both S1 layers – the C1 cell just takes a max over any cell in its receptive field.

In FH models, the dimensions Y and X correspond to retinal position and are meaningful across all layers. In CN, the same holds in the $X_i$ layers for spatial dimensions $Y$, $X$, and $D$ (depth). Thus, cells in both packages are assigned to real-valued grid positions in these common dimensions. Fig. A-6 illustrates how these coordinates are used to perform an operation (pooling over scales that are not an integer multiple of one another in an FH model) that is difficult to define without them.

Note that this is just one instance of a general problem which can arise in several ways, and which CNS’s common coordinate system was designed to solve: how does one define 'local' operations like convolution over multiple layers having different sizes, resolutions, etc.? Networks having multiple pathways that converge will almost always have this issue, since the different pathways probably involve different amounts of convolution (with edge loss) and/or subsampling. And of course, convergence
occurs throughout cortex.

Part of defining a network model in CNS is setting up these grid positions for the different layers. There is a function (\texttt{cns\_mapdim}) that handles all the standard cases: valid, same, and invalid convolutions, subsampling, supersampling, etc. Once grid positions have been assigned, connectivity is handled by CNS \texttt{U-} in kernel code, you simply call a macro requesting, for example, the nearest $n$ neighbors in some layer $z$, or every cell within radius $r$.

Not all dimensions in a model need to have a common coordinate system. For example, the non-spatial dimensions in FH and CN models do not have the notion of locality, so for them this scheme is not used.

A.4 Working with CNS

In this section we provide a quick overview of the process of developing and running CNS models.

Before you can build a network model, you must define your cell types. This is done by creating a \textit{package}, which is a collection of related cell types, or by acquiring and possibly modifying an existing package. For example, if you wanted to build a network of Hodgkin-Huxley spiking cells, you would probably want to start with the existing HH package. The properties of cell types and their associated fields and kernels are described in \S A.3.3, \S A.3.4, and \S A.3.5. Most of these definitions are made in MATLAB '.m' files, with kernels in C/C++ '.h' files. All such files for a single package are stored together in a package directory. Before you can use a package, it must be compiled into an executable using the \texttt{cns\_build} command. This actually produces two executables, one that runs models on a GPU and another that runs models on a CPU. The latter option can be useful for development and debugging.

Once you have a compiled package, you can use it to build and run any number of different network models, having different numbers and sizes of layers, with different connectivity, etc. The process of building and running a network model is as follows.
All steps are carried out from within MATLAB using a few CNS commands.

1. Define the structure of your network: the number of layers, their sizes and types, cell-to-cell connectivity, and the initial values of fields. This is all defined in a MATLAB struct which you create. The helper function cns_mapdim can assist you in setting up a common coordinate system (§ A.3.6).

2. Initialize the model on the GPU (or CPU). This can take a few seconds.

3. Run the model for any number of iterations.
   - For dynamic models (e.g. HH models) this may involve loading some input data and then letting the model iterate for awhile. Variables can be tracked as they change, or if you just want their final values, you can pull them out when all iterations are complete.
   - Other models (e.g. FH models) process their input in a single iteration. For these models you simply load your input data, perform a single iteration, and pull out any output data. For batch operations (e.g., processing many images with an FH model) you would perform those three operations inside a loop.

4. Deallocate the model, freeing resources.

### A.5 GPU details

While there are some circumstances in which model choices have performance consequences – these are documented in the programmer’s manual [28] – users in general do not need to understand GPU programming in order to use CNS.

Here we provide a few details on how CNS works behind the scenes. This section is intended for those familiar with GPU programming concepts; an exposition of GPU programming and architecture is beyond the scope of this report.

**Dimension mapping.** The user sees layers as N-dimensional arrays, and this ab-
straction is consistent throughout CNS: both MATLAB functions and kernel macros view layers as N-D. However, internally layers are stored as 2-D. This is done because some fields need to be stored in GPU textures (see below). Textures can be 2-D or 3-D, but for 3-D textures the size limits are prohibitively small as of this writing. It is also more difficult to do texture packing in 3-D. Thus, CNS uses only 2-D textures.

All translation between N-D and 2-D is handled by CNS. The user’s only involvement in this process occurs when defining a cell type. The user must choose the internal dimension (Y or X) to which each external dimension is mapped. Current GPU limits on texture size will influence these decisions. When two or more external dimensions are mapped to the same internal dimension, the user must also specify the nesting order. This choice has performance consequences. Storage will only be contiguous for the innermost dimension, so kernels involving nested loops should iterate over the innermost dimension in the innermost loop. (This is done in fig. A-5.) CNS automatically pads the innermost Y dimension to the warp size so that warps will perform coalesced reads as often as possible.

Types of GPU memory. CNS automatically maps each field to the appropriate kind of GPU memory based on the field’s definition:

- Small, shared constants having layer, group, or model scope are stored in the constant cache. The constant cache is also used to store internal metadata such as layer sizes.
- Fields that have cell scope and that will be read by other cells are stored in textures. One texture is used for all the layers having that field; texture packing is done automatically.
- Everything else goes into global memory.
- Shared memory is not used.

Blocks and threads. All layers having the same cell type are evaluated in a single kernel call (unless the user has assigned them to different step numbers; see § A.3.2).
The decomposition into blocks and threads is done automatically. Each cell gets its own thread. Each layer becomes one or more thread blocks; an individual thread block will contain threads from a single layer only. Blocks are 2-D and aligned in the same way as memory (see above) to maximize the number of coalesced reads and the benefits of texture caching.

A.6 Limitations

A.6.1 Inherent limitations

The following limitations are inherent in CNS’s role as a rapid development framework for expressing arbitrary models having a cortical organization:

- CNS implements a generic, automatic process for mapping cortical models to the GPU architecture. For any given specific model, it will always be possible for a sufficiently skilled programmer to write custom GPU code that runs faster by taking advantage of optimization techniques peculiar to that model. However, custom GPU code is much harder to write and modify. The speedups we are seeing under CNS relative to a single CPU (80-100x) are on the order of what is typically reported in the literature for direct GPU implementations of various algorithms. Our testing so far suggests that, at worst, a CNS model will be no more than 2x slower than a carefully-written custom GPU implementation of that specific model.

- Models must run inside a single GPU. Host-GPU communication is relatively slow, and CNS makes no assumptions about the sparsity of long-range vs. local connectivity, nor about the frequency of cell-to-cell communication. A less general framework in which such assumptions could be made might be able to automatically decompose models into pieces that could run on separate GPUs without incurring prohibitive data transfer delays. CNS cannot do this, so any such decomposition must be done by the user, outside CNS. Barring a dramatic improvement in the speed of host-GPU communication, this limitation cannot
be removed, although it might be possible to implement automatic solutions for a subset of cortical models.

A.6.2 Current limitations

These limitations could potentially be removed, some more easily than others:

- Only NVIDIA cards are currently supported.
- You cannot define fields that store 64-bit quantities, such as double-precision floating point numbers. Allowing fields of different sizes would somewhat complicate the current CNS code. Note that temporary variables used inside kernels can be 64-bit now.
- CNS is MATLAB-dependent. Since most of CNS is written in MATLAB, porting it would be a big job.
- Limitations on the size of the 2-D textures (currently 64K x 32K) used to store some fields can sometimes complicate model definition. This is beyond our control; however, given the current industry push to open up GPUs for general-purpose computing, it does not seem unreasonable that this limit might be removed in future cards.

A.7 Summary of contributions

CNS was one of the first (and is still the most flexible) GPU-based software frameworks for the rapid development of cortically-organized computational models. Prior to CNS, GPU implementations were typically hard-coded translations of models which were the end result of experimentation carried out on slower hardware. GPU implementations were inflexible, discouraging exploratory work on that platform.

CNS is publicly available, together with an online manual [28]. There have been a number of releases, adding support for multiple GPUs and streaming video, among other enhancements, and maintaining compatibility as newer GPUs were released.
At this writing CNS has been downloaded by 1,058 unique individuals from 72 countries. It has been used in many projects with no direct involvement by the author. Examples found via Google searches include [25, 14].

The author personally contributed to the following BCS (or affiliated) projects:

- **Connectomics (Seung lab)** - wrote first version of a CNS package for convolutional backprop in 3 spatial dimensions. Supported Seung lab members in training networks for automatic segmentation of cell bodies in 3D EM stacks. [16, 10, 20]

- **Action recognition in video for mouse behavioral phenotyping (CBCL)**. [17]

- **General action recognition in video (CBCL)** - provided extensive advice and support in creating the initial CNS package for this project. [48]

- **Action recognition in video (Google Research)** - assisted Tom Dean in development of models, some described in [4].

- **Large spiking network simulations of realistic neurons (CBCL / Moore lab)**. [21]

- **Intelligent parameter searching over HMAX model architectures (CBCL / IIT)**. (chapter 13 of [7])

- **Object recognition from ariel video (CBCL / USC / etc.)** - powered the core object recognition model used in the MIT/USC team’s entry for the DARPA Neovision 2 challenge. [19]

CNS has also been used for action recognition by Tom Dean at Google Research, with some help from the author. CNS was the subject of his talk at NVIDIA’s GPU Technology Conference in 2010.

Most of the work described in this thesis has been carried out using CNS.
A.8 Future work

With CNS stable, our main focus is now on specific package and model development. Likely future improvements to CNS itself include:

- Taking advantage of some of NVIDIA's more recent GPU features. Two changes of relevance to CNS are the ability to run more than one kernel concurrently and the option to enable L1 caching.

- Investigating the feasibility of supporting non-NVIDIA cards.

Acknowledgements

CNS began during a collaboration with Ulf Knoblich on a project modeling networks of spiking neurons. Ulf wrote the initial HH package and provided feedback, suggestions, and enthusiasm, all in large quantities.

We thank the early adopters of CNS for their helpful feedback: Hueihan Jhuang, Sharat Chikkerur, Srini Turaga, Kannan Venkataraju, Matt Greene, and Viren Jain.
Bibliography


[27] Jim Mutch. FHLib Homepage.


