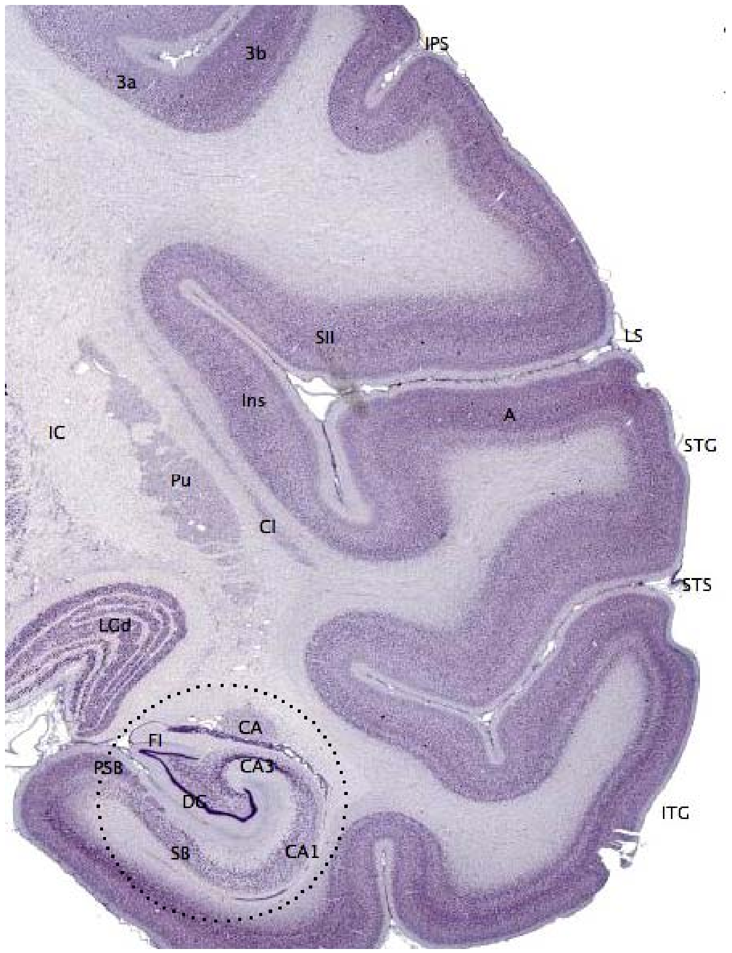
GPGPU-Based Cortical Modeling

Brains on a Budget

Ted Hilk

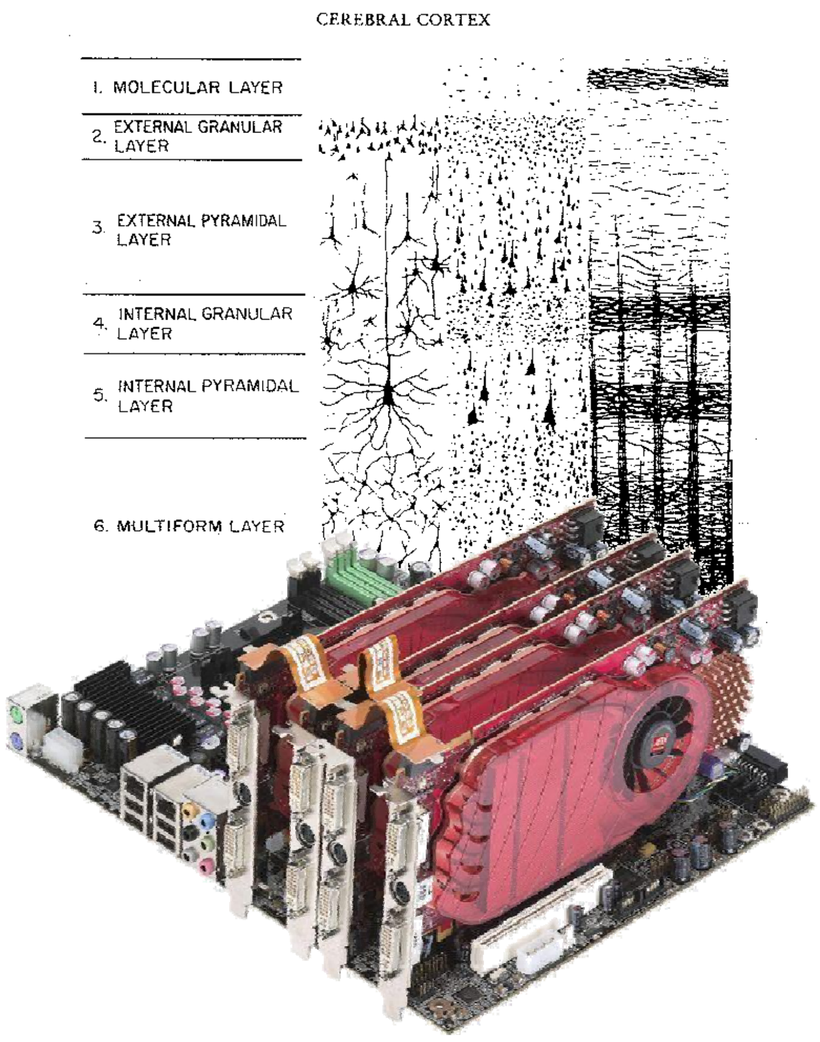
Cortical Modeling Overview

- Modeling and simulating the cerebral cortex
- Why?
  - Center of memory, attention, language, conscious thought….
- Goals:
  - Understand experimental neuroscience data better
  - Build improved AI systems
- Challenges:
  - Model accuracy
  - Model generality
  - **Computational power**



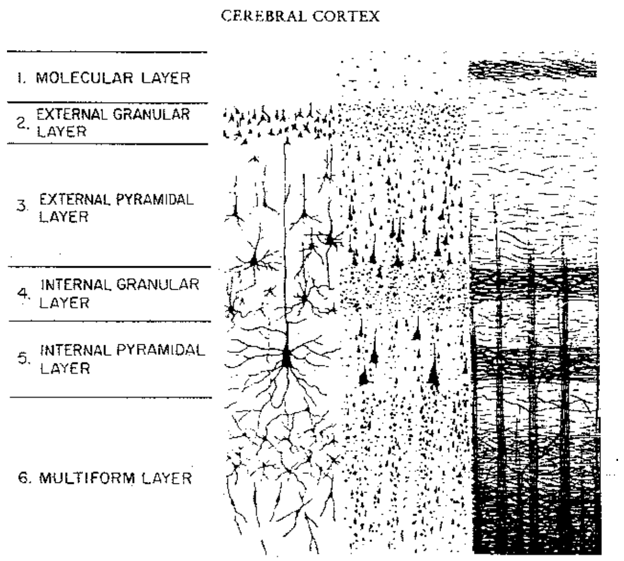
Project Rationale

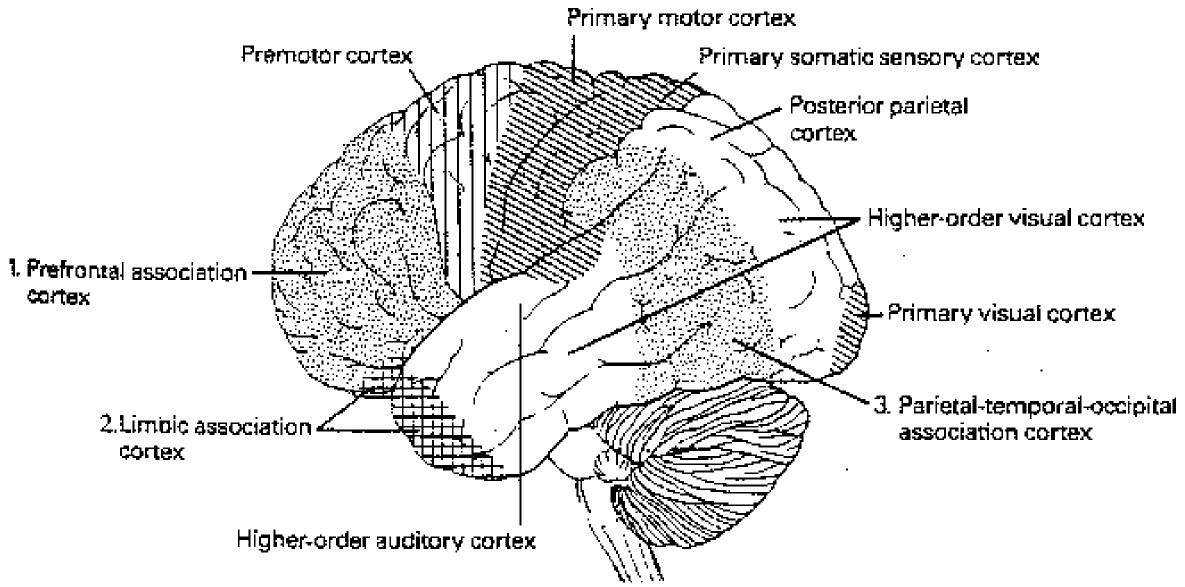
- Problems:
  - Current cortical modeling approaches omit key cortical functionality
  - Current model frameworks lack vendor neutrality
- Solution:
  - Create a vendor-neutral GPGPU framework for cheap access to high-performance computing
  - Use it to work on new cortical models (6.UAP)


CEREBRAL CORTEX
1. MOLECULAR LAYER
2. EXTERNAL GRANULAR LAYER
3. EXTERNAL PYRAMIDAL LAYER
4. INTERNAL GRANULAR LAYER
5. INTERNAL PYRAMIDAL LAYER
6. MULTIFORM LAYER

Cortical Modeling: Themes

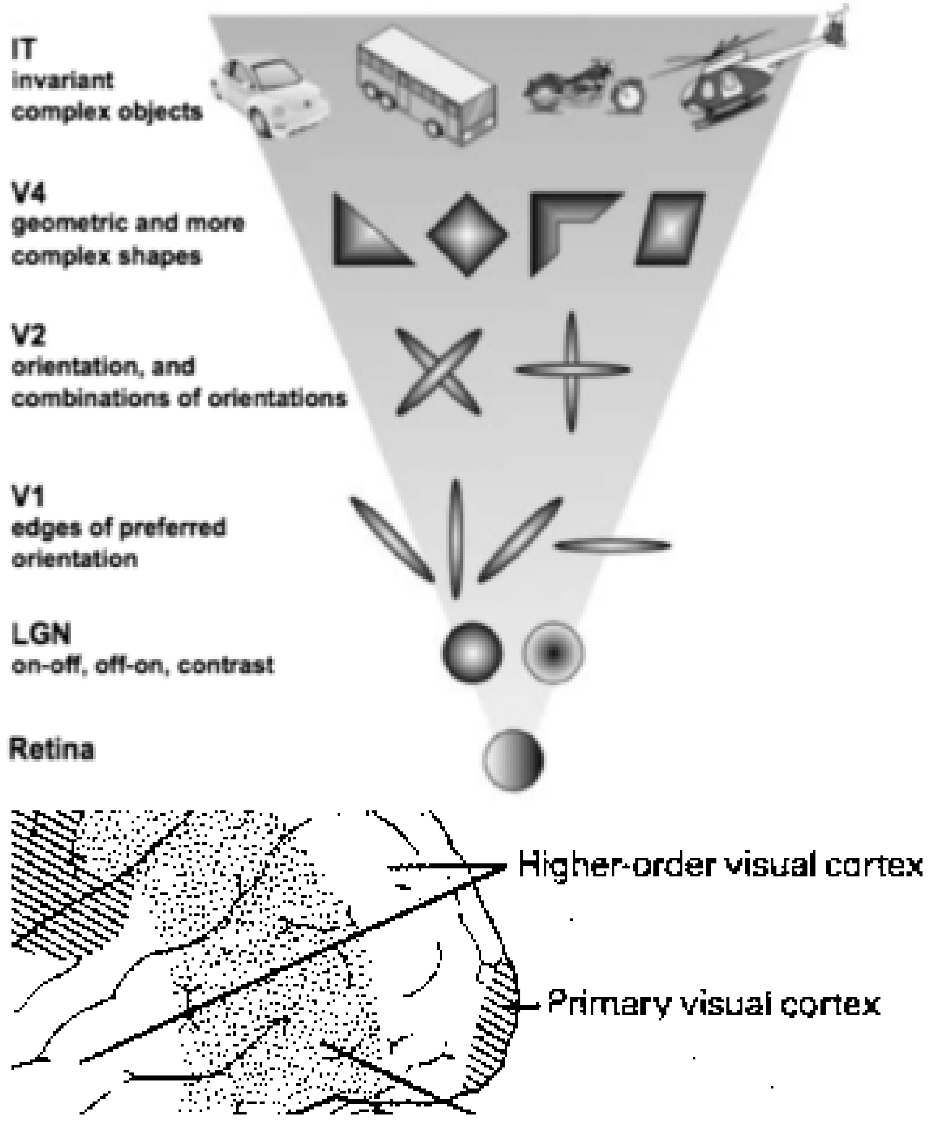
- **Hierarchy**
- Spatial clustering
- Temporal clustering
- Feedback
- Prediction
- Dimensionality Reduction
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance


CEREBRAL CORTEX
1. MOLECULAR LAYER
2. EXTERNAL GRANULAR LAYER
3. EXTERNAL PYRAMIDAL LAYER
4. INTERNAL GRANULAR LAYER
5. INTERNAL PYRAMIDAL LAYER
6. MULTIFORM LAYER


Primary motor cortex
Premotor cortex
Primary somatic sensory cortex
Posterior parietal cortex
Higher-order visual cortex
1. Prefrontal association cortex
Primary visual cortex
3. Parietal-temporal-occipital association cortex
2. Limbic association cortex
Higher-order auditory cortex

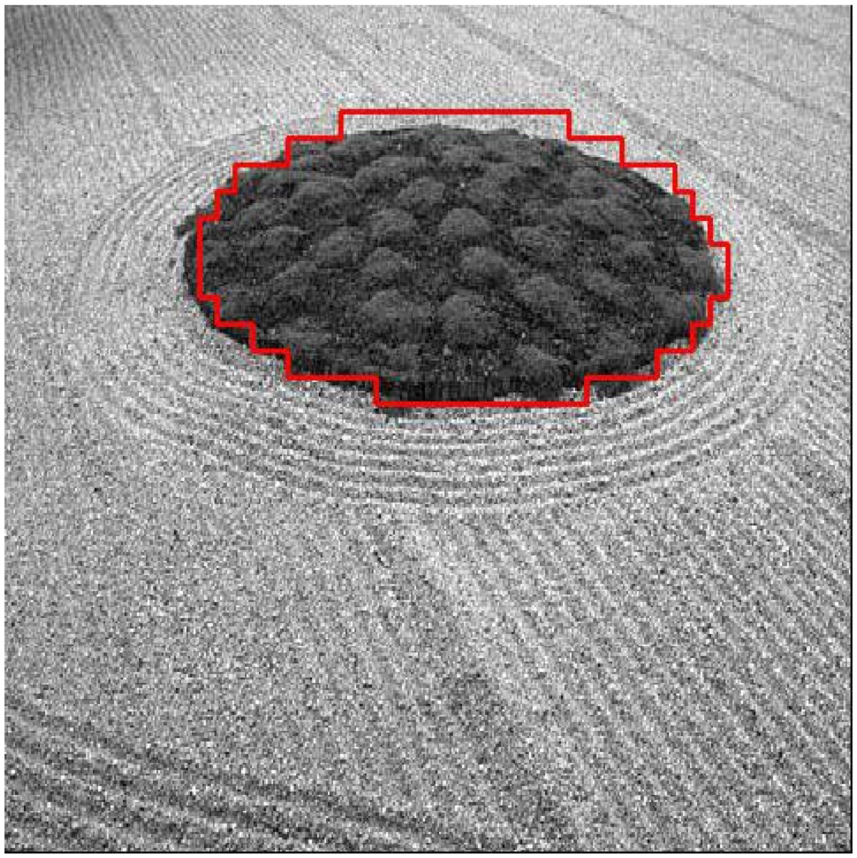
Cortical Modeling: Themes

- **Hierarchy**
- Spatial clustering
- Temporal clustering
- Feedback
- Prediction
- Dimensionality Reduction
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance


IT
invariant complex objects
V4
geometric and more complex shapes
V2
orientation, and combinations of orientations
V1
edges of preferred orientation
LGN
on-off, off-on, contrast
Retina
Higher-order visual cortex
Primary visual cortex

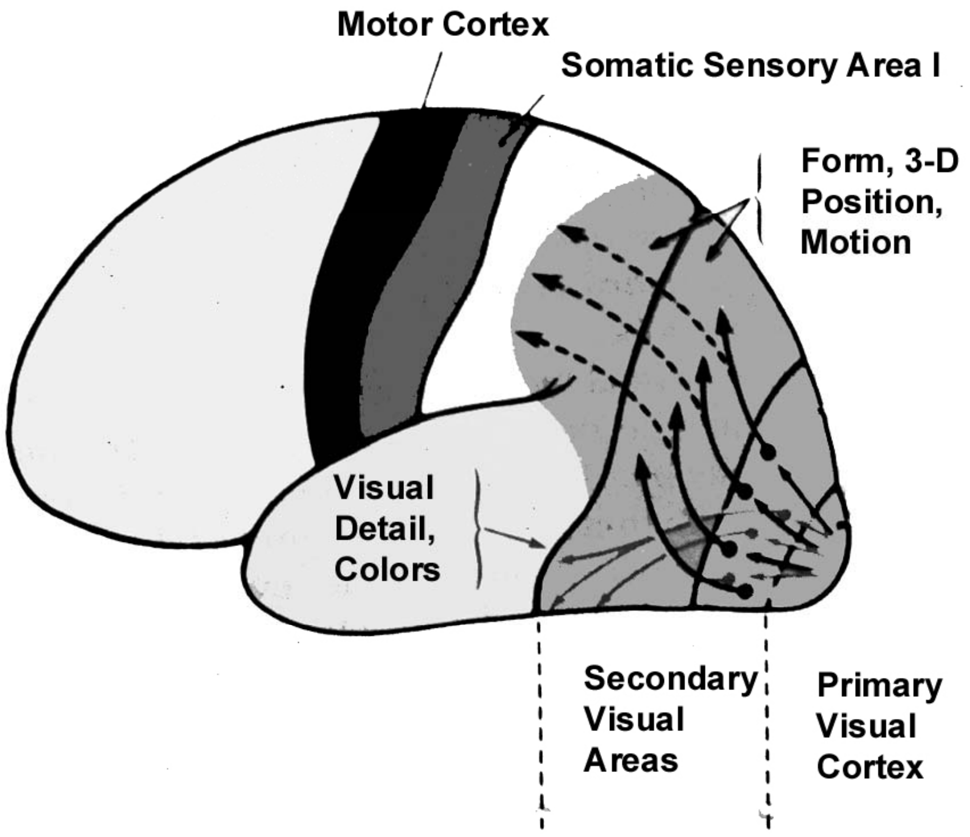
Cortical Modeling: Themes

- Hierarchy
- **Spatial clustering**
- Temporal clustering
- Feedback
- Prediction
- Dimensionality Reduction
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance



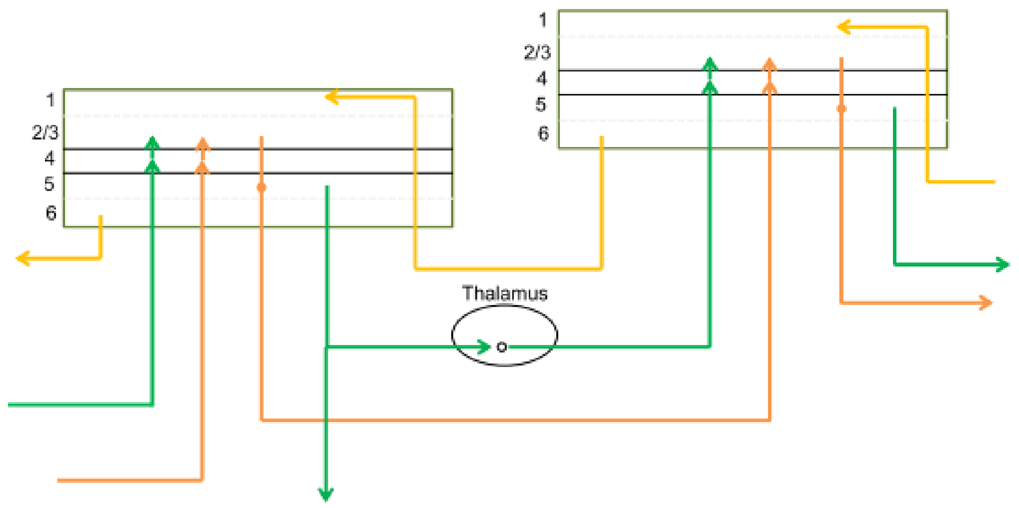
Cortical Modeling: Themes

- Hierarchy
- Spatial clustering
- **Temporal clustering**
- Feedback
- Prediction
- Dimensionality Reduction
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance



Cortical Modeling: Themes

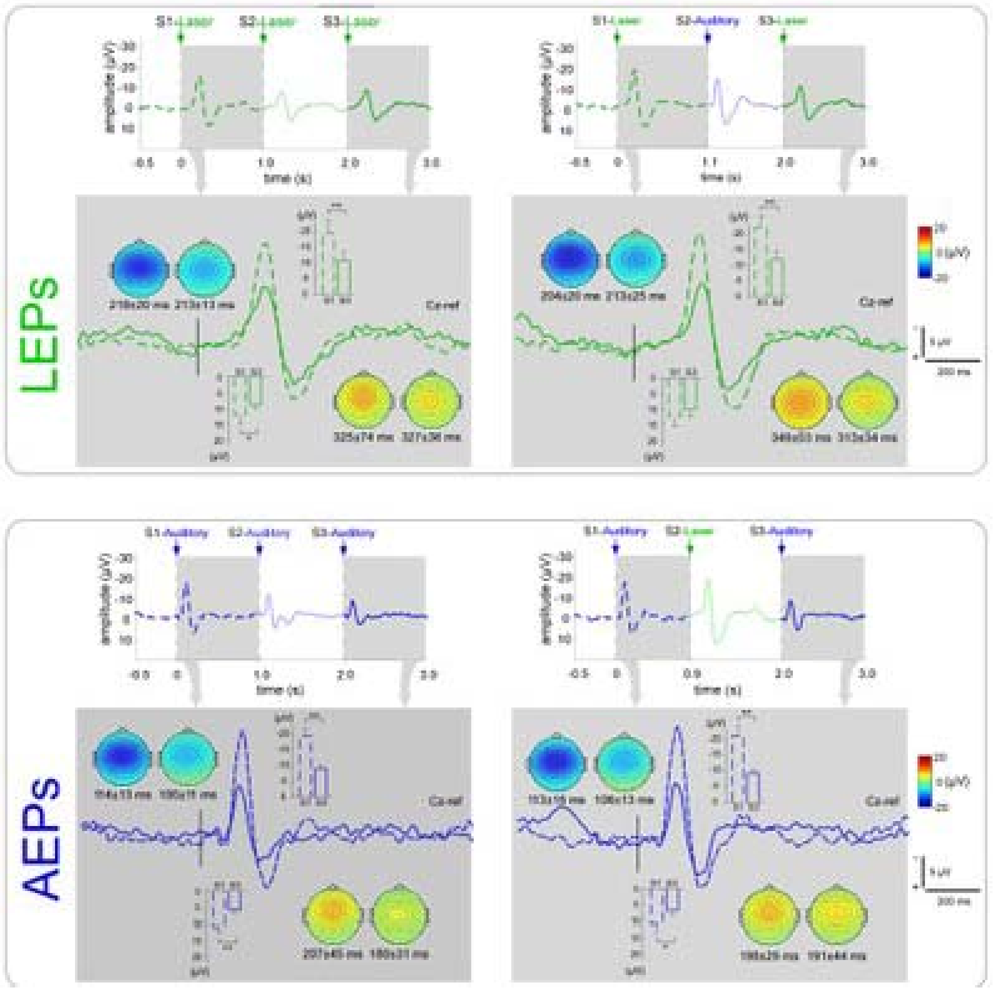
- Hierarchy
- Spatial clustering
- Temporal clustering
- **Feedback**
- Prediction
- Dimensionality Reduction
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance



Schematic representation of cortical connectivity between regions

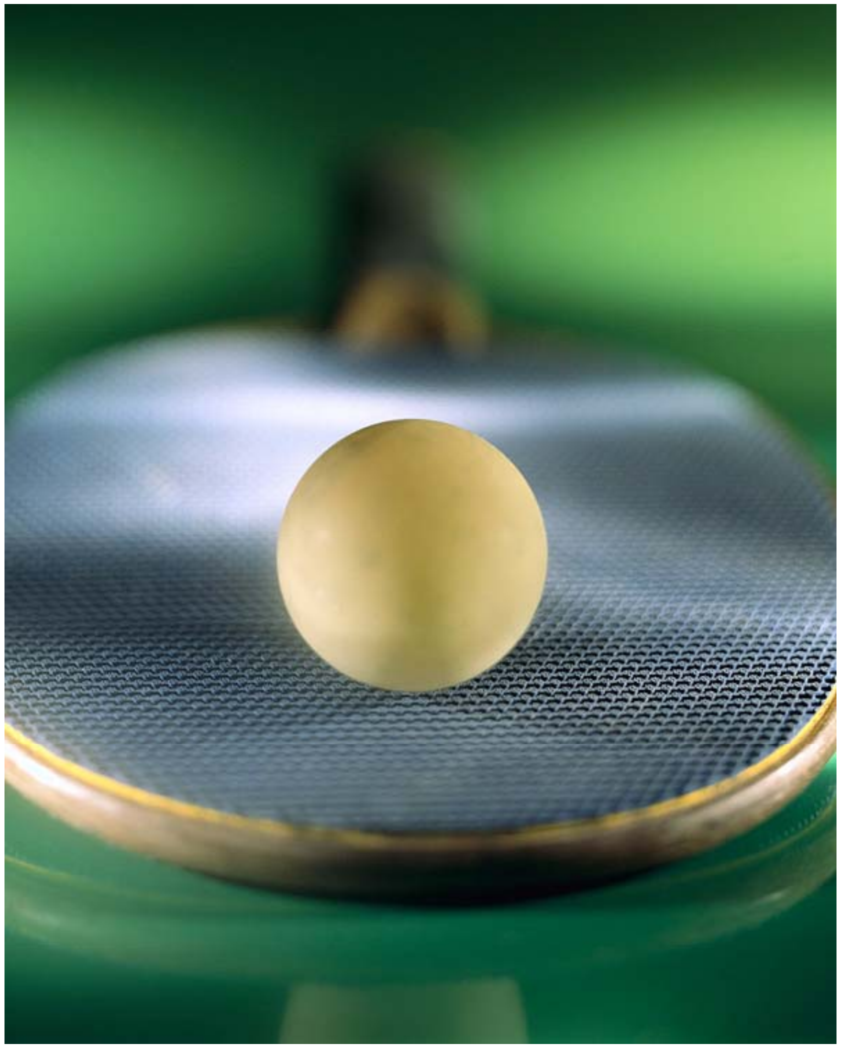
Cortical Modeling: Themes

- Hierarchy
- Spatial clustering
- Temporal clustering
- Feedback
- **Prediction**
- Dimensionality Reduction
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance



Cortical Modeling: Themes

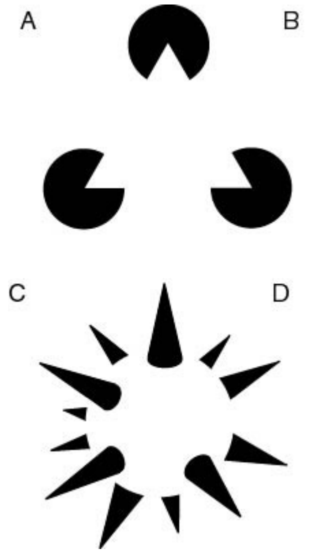
- Hierarchy
- Spatial clustering
- Temporal clustering
- Feedback
- Prediction
- **Dimensionality Reduction**
- Gestalt Principles
  - Emergence
  - Reification
  - Multistability
  - Invariance

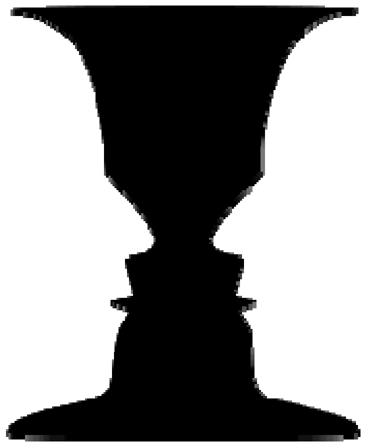


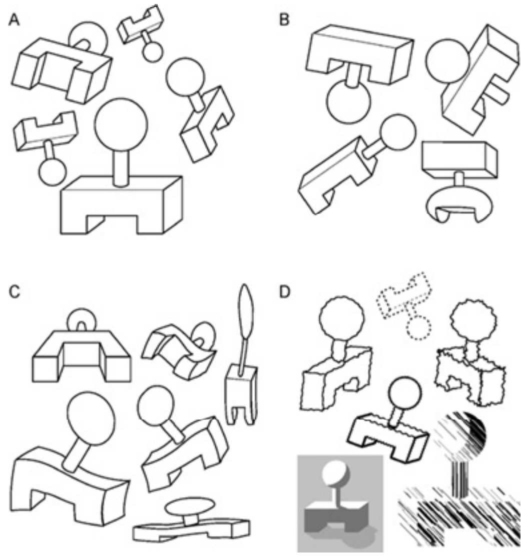
~218,400 $\rightarrow$ 3

Cortical Modeling: Themes

- Hierarchy
- Spatial clustering
- Temporal clustering
- Feedback
- Prediction
- Dimensionality Reduction
- **Gestalt Principles**
  - **Emergence**
  - **Reification**
  - **Multistability**
  - **Invariance**







HMAX

- Poggio, Serre (MIT CBCL)
- Model specifically for visual cortex
- Alternating layers of S (selectivity) and C (combination) units, with MAX operations in the C units to pool over inputs
- Includes human-created features
- Performance on par with the best computer vision systems; rivals human evaluators

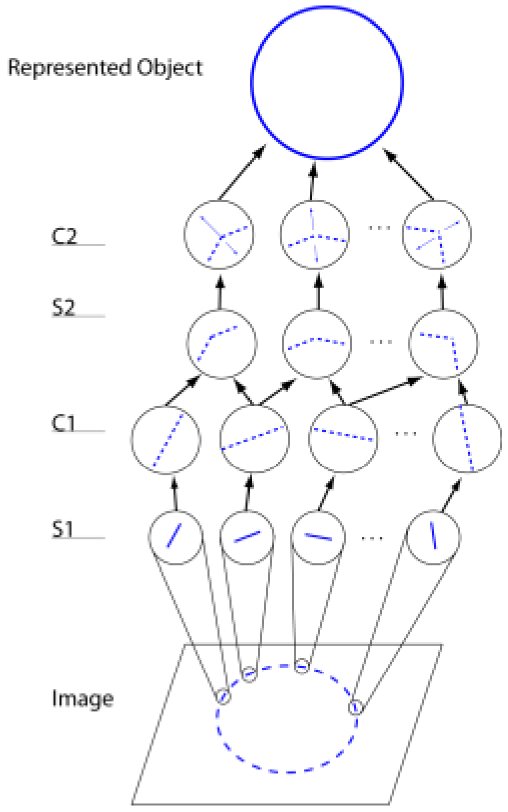
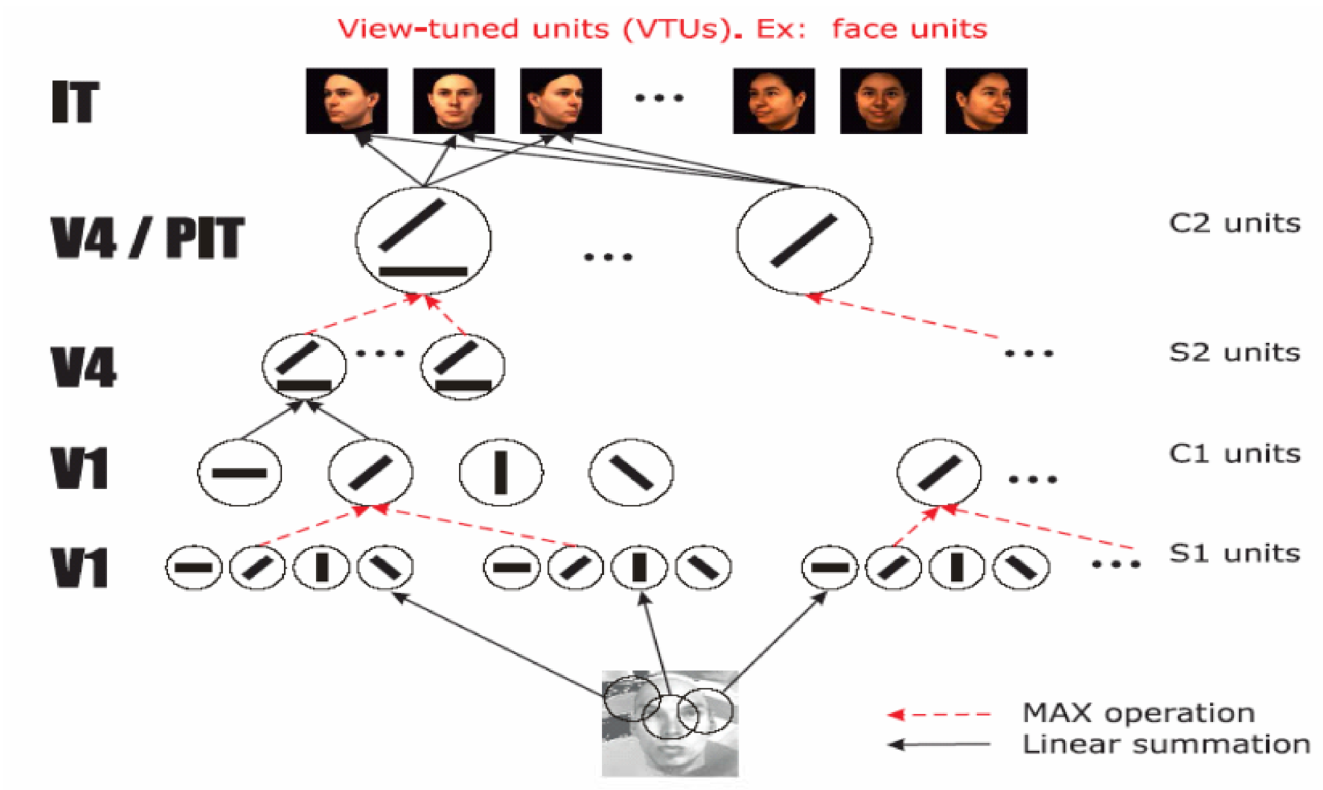


Figure 1: The HMAX model.

HMAX

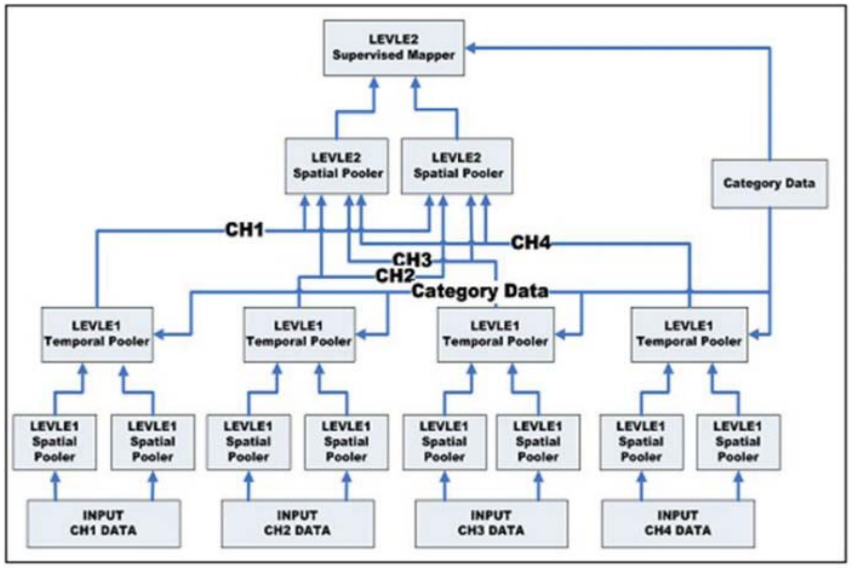


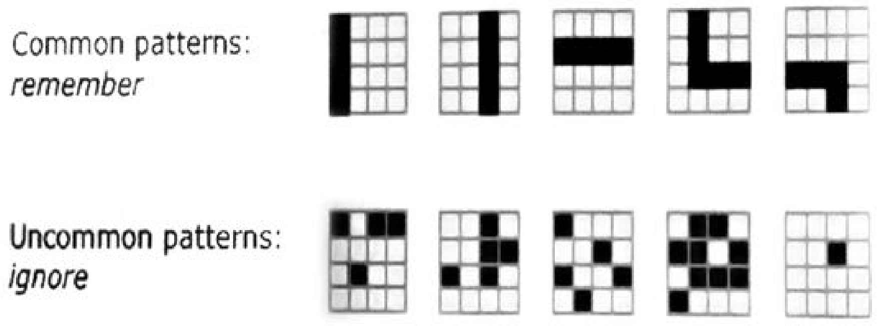
Serre, T., and Riesenhuber, M. (2004)

Alternating layers of S (selectivity) and C (combination) units, with MAX operations in the C units to pool over inputs

HTM

- George, Hawkins (Numenta)
- Alternating layers of temporal and spatial poolers
- Complex, pseudo-biological “neurons” with specific connectivity and activation properties
- Sequence-based local cell memories
- Generality can be poor; learned results often “brittle”

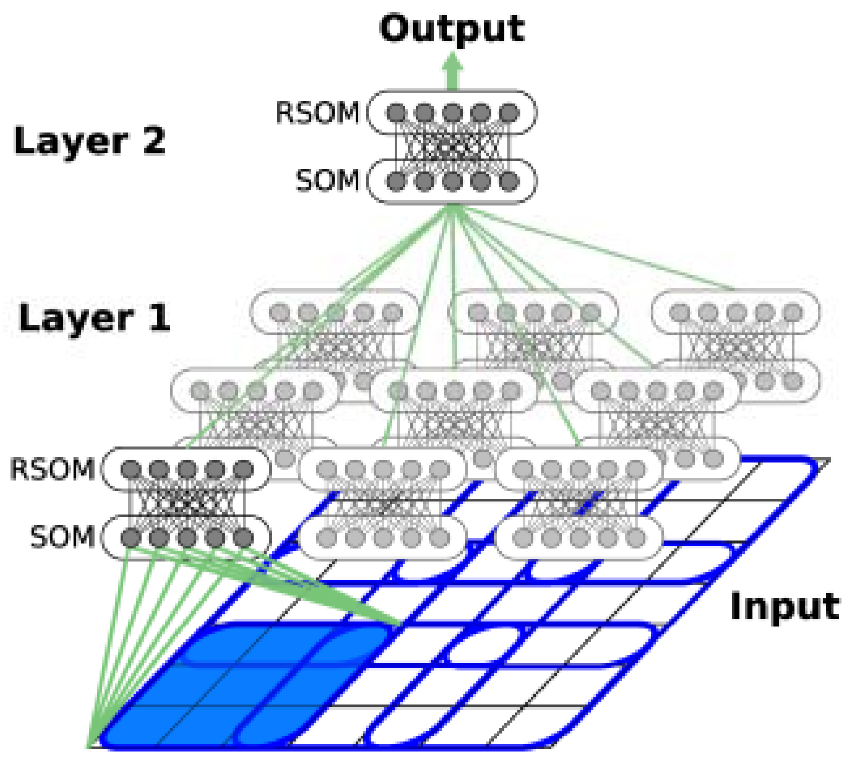




Remembering spatial patterns over a length of time.
Most common patterns are remembered.

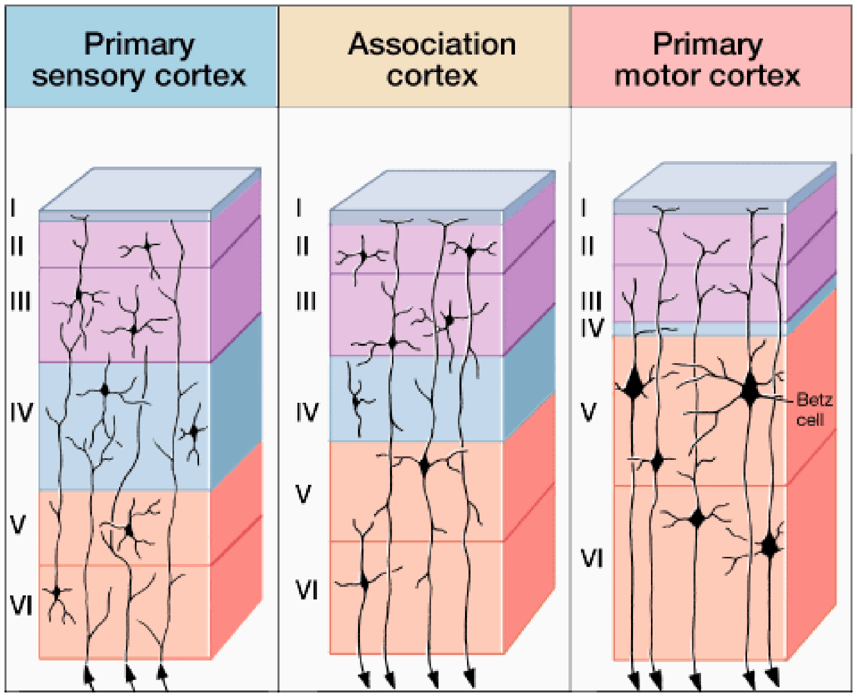
HQSOM

- Miller, Lommel (Draper)
- Hierarchical quilted self-organizing map
- Comprised of pairs of self-organizing maps (SOMs) wired into recurrent SOMs (RSOMs).
- SOM handles spatial feature representation
- RSOM handles temporal features, i.e. transitions
- More general than HMAX; less complex than HTM



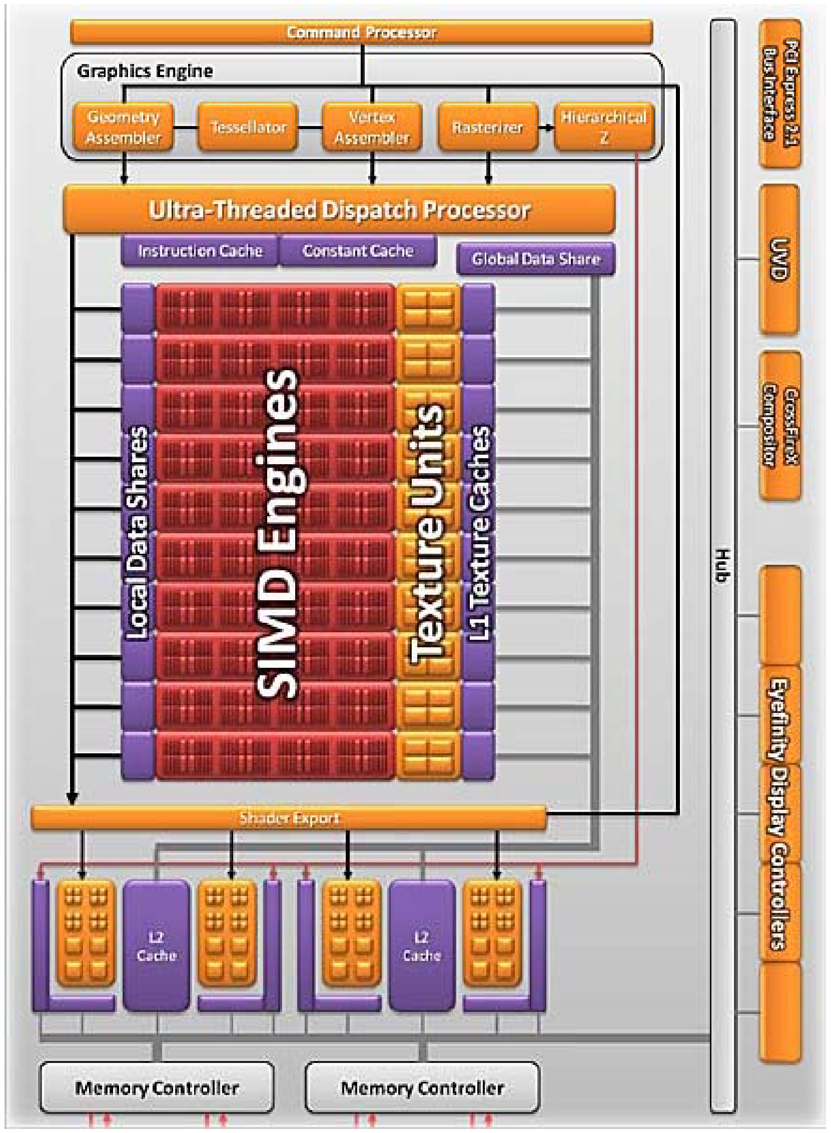
Gaps in Current Models

- Frequently not on-line
- Inadequate emphasis on feedback and prediction
- Lack of motor output
  - Corresponding lack of operant learning
- Absence of reflexes
- Lack of attention regulation
- Lack of emotional responses
- Absence of clarity regarding episodic memory storage
- **Do not account for heterogeneities in cortical structure**
- *Active areas of research!*



GPU Supercomputing

- Cheap teraflops from commodity graphics cards
- Massively SIMD
- General-purpose parallel programming libraries
  - NVIDIA: CUDA
  - ATI: CAL/Brook+/Stream
  - **Open standard: OpenCL**
- Extensive developer communities
- Applications in physics, biology, finance….



Existing GPGPU Cortical Modeling Frameworks

- Very few examples, e.g.:
  - CNS – Poggio 2009
  - Nere et. al. 2012
- All previous frameworks based solely on CUDA
  - Vendor lock-in – NVIDIA
  - Particularly unfortunate; ATI cards often superior for compute-bound workloads
- Existing tools also impose extra model assumptions/ overhead

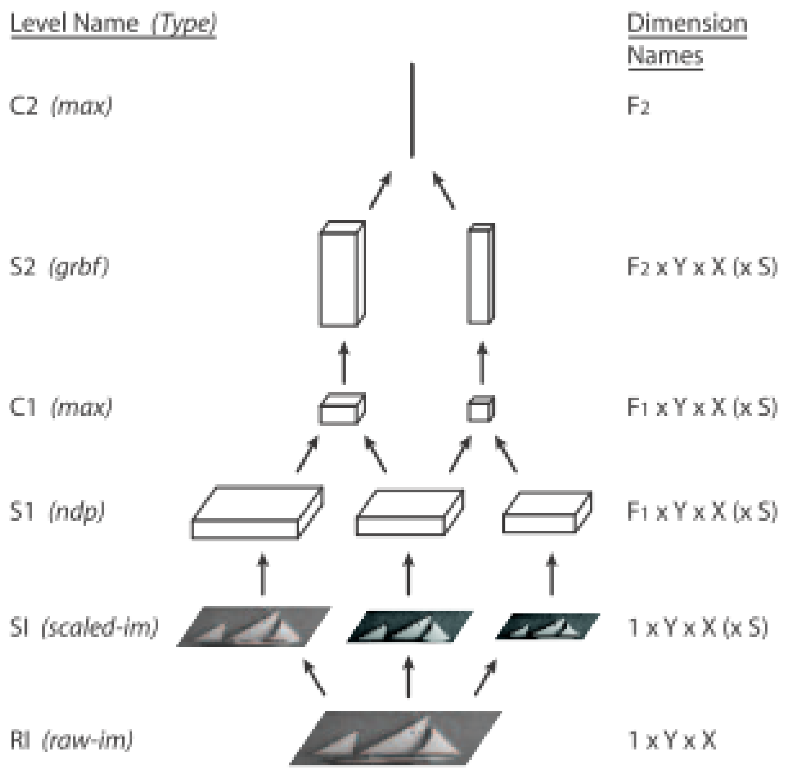
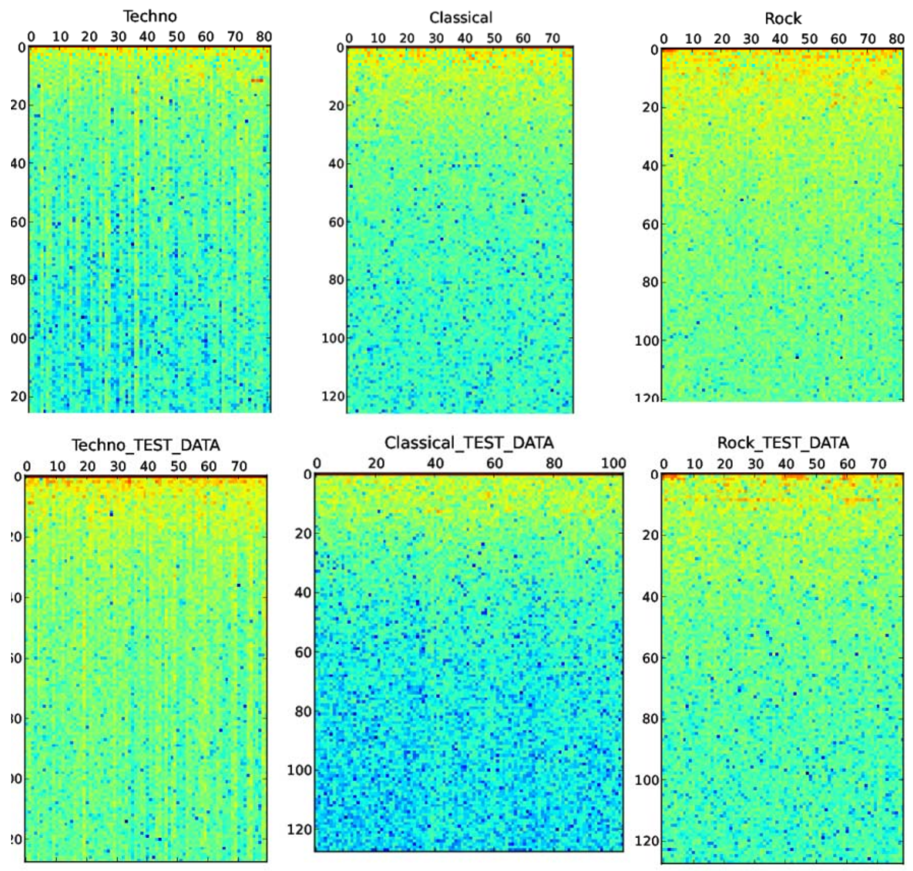


Figure 3. An example of an HMAX model (see section 2.2). Each step is performed at multiple scales, only three of which are shown here.

CNS network for an example HMAX model

Prior Work

- 6.867 final project
  - Implemented HQSOM in Python (single-threaded)
  - Tweaked algorithm based on observed performance/limitations
  - Replicated authors' results on visual classification
  - Wrote spectrogram wrapper to create successful audio classifier

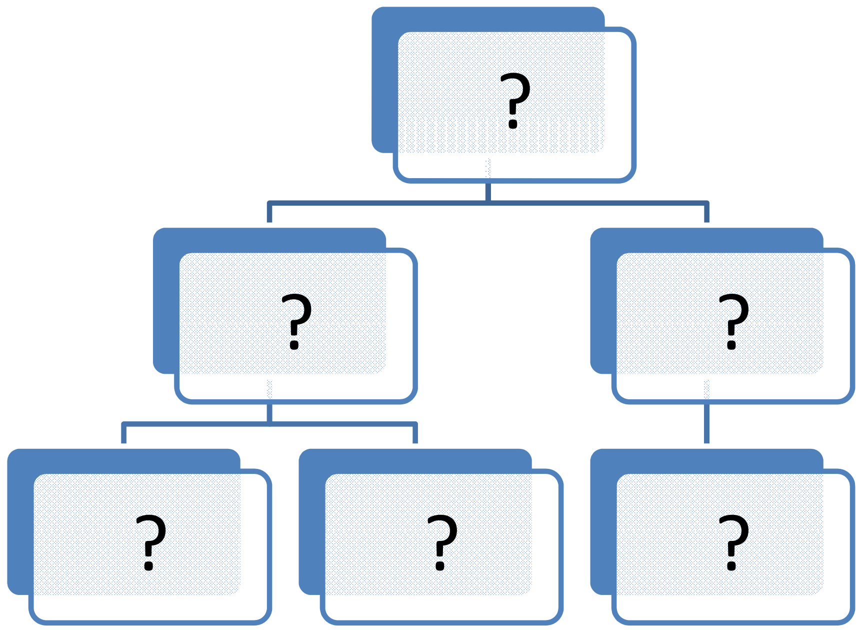


Spectrograms used by the audio classifier

Project Overview

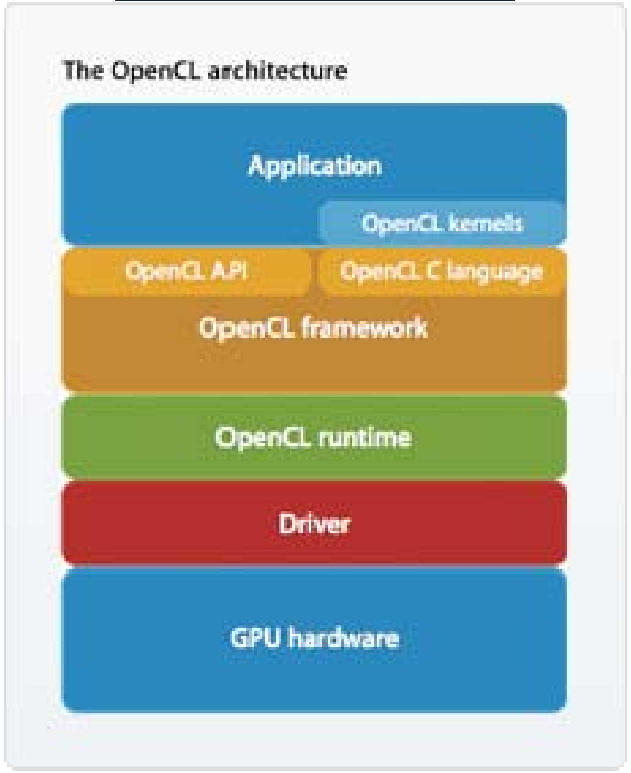
- Implement OpenCL GPU framework for general network cortical models
- Implement HQSOM using this framework
- Further extend HQSOM to automate parameter tuning, time permitting and later for 6.UAP
- Try other clustering methods later for 6.UAP

Your Model Could be Here!



GPU Framework

- Model Implementation
  - Node state: on-GPU buffers
  - Algorithms: OpenCL kernels
  - Inter-node i/o: specific i/o buffers
  - Everything stays on GPU!
- Layout Manager
  - Nodes, regions, networks
  - Feedforward and feedback links
- Execution Manager
  - Node "execute()" method fetches input, runs kernels, caches output
  - Region-wise execution – no need for ordering guarantees
  - Appropriate use of barriers for synchronization with minimal performance impairment
  - Manage GPU resources and scheduling with PyOpenCL!



PyOpenCL

or: “How I learned to stop worrying and love the GPU”

- (*Edited from the website*–) PyOpenCL lets you **access the OpenCL parallel computation API from Python**. Here's what sets PyOpenCL apart:
  - **Object cleanup tied to lifetime of objects.** This idiom, often called RAII in C++, makes it much easier to write correct, leak- and crash-free code.
  - **Completeness.** PyOpenCL puts the full OpenCL API at your disposal, if you wish.
  - **Convenience.** While PyOpenCL's primary focus is to make all of OpenCL accessible, it tries hard to make your life less complicated as it does so—without shortcuts.
  - **Automatic Error Checking.** All OpenCL errors are automatically translated into Python exceptions.
  - **Speed.** PyOpenCL’s base layer is in C++, so the niceties above are virtually free.
  - **Helpful, complete documentation** and a wiki.
  - **Liberal licensing (MIT).**

PyOpenCL

or: "How I learned to stop worrying and love the GPU"

Blazing-fast parallel infrastructure

+

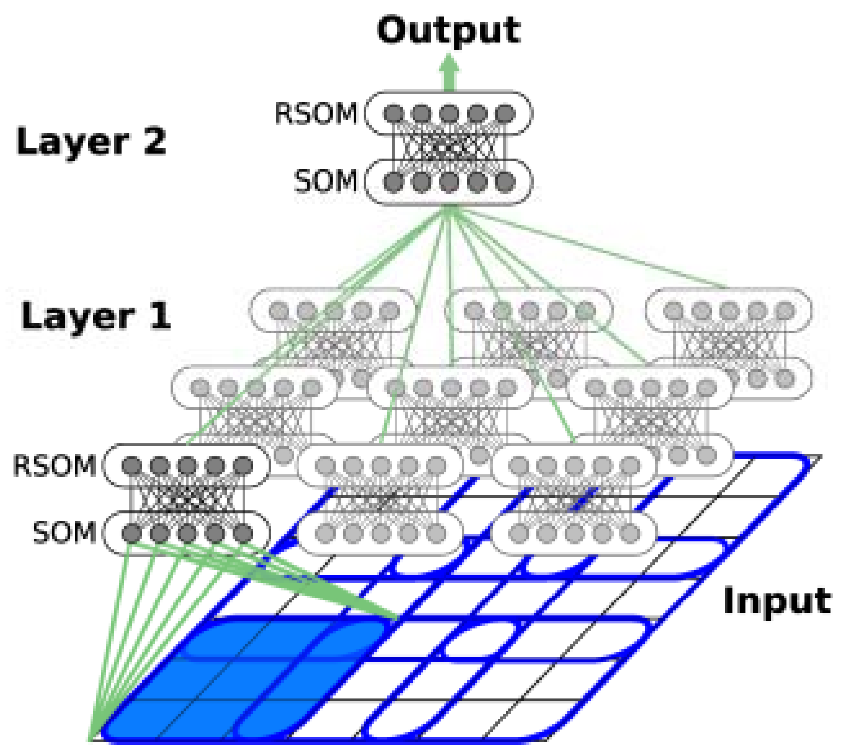
Convenient management scripting

=

WIN

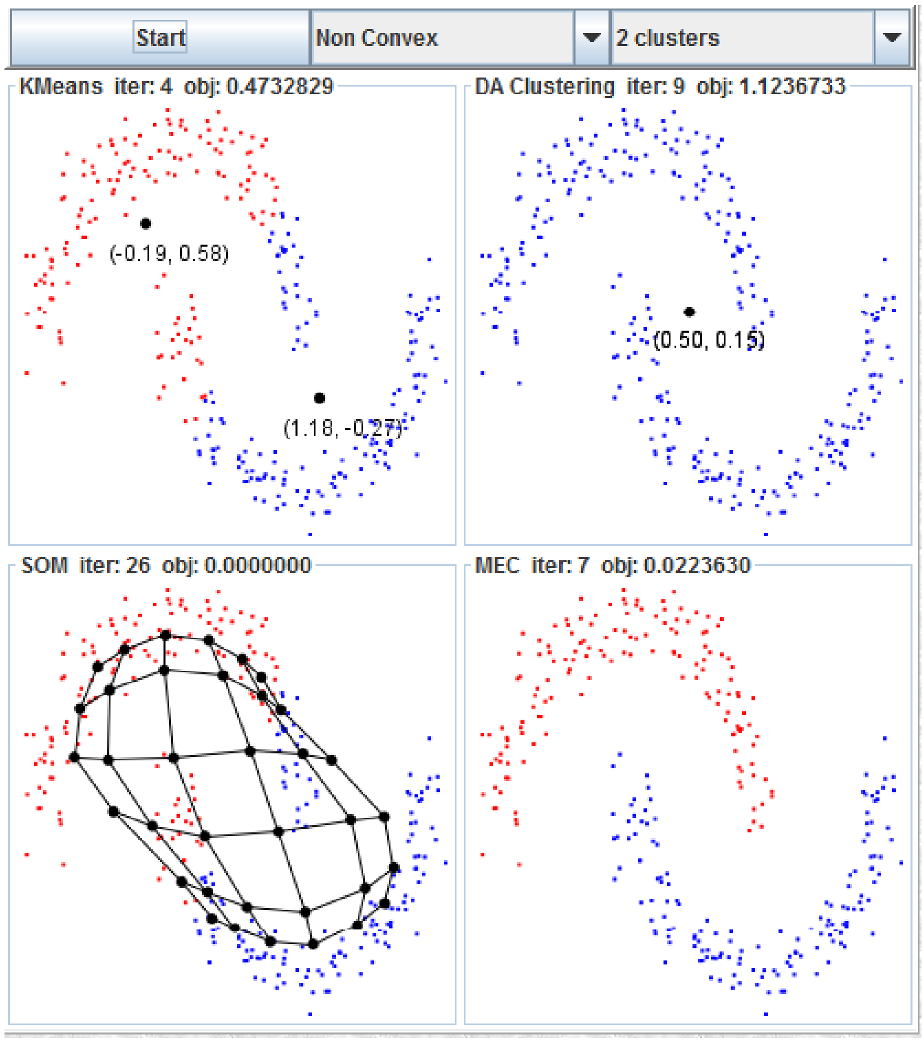
HQSOM (Re-)Implementation

- Pre-existing Python code for HQSOMs from 6.867 final project
- Must port to a set of OpenCL kernels and intermediate buffers
  - Everything else handled by node superclass
  - i.e. only need to port the SOM-RSOM pair code
- Then simply plug nodes into network framework



Potential Model Extensions

- Automate parameter tuning for the HQSOM
  - adaptive sigma
  - adaptive gamma
- Investigate other models, clustering techniques beyond SOMs
  - density-based clustering
  - information-theoretic clustering approaches
  - fuzzy extensions thereof
  - deep belief networks?


Start
Non Convex
2 clusters
KMeans iter: 4 obj: 0.4732829
(-0.19, 0.58)
(1.18, -0.27)
DA Clustering iter: 9 obj: 1.1236733
(0.50, 0.15)
SOM iter: 26 obj: 0.0000000
MEC iter: 7 obj: 0.0223630

Conclusion

- Implementing a vendor-neutral GPGPU solution for cortical modeling
- Cortical modeling well-suited to massive SIMD parallelism of GPGPU platforms
- Substantial computational speed-up permits faster simulations/trials, larger networks
- Implications for both neuroscience and biologically-inspired artificial intelligence