Unlocking Biologically-Inspired Computer Vision: a High-Throughput Approach

Nicolas Pinto, David Cox and James DiCarlo
NVIDIA GTC | October, 2009

The Rowland Institute at Harvard
Unlocking Biologically-Inspired Computer Vision: 
a High-Throughput Approach
Unlocking Computer Vision: a High-Throughput Approach

= \text{Brain (Neurosciences)}
Unlocking Biologically-Inspired Computer Vision: a High-Throughput Approach

A.I.
Unlocking Biologically-Inspired Computer Vision: a High-Throughput Approach
Friend: So, what are you studying for your PhD?

Me: I study biological and artificial vision.

Friend: What?!? But vision is super easy!
The Problem:
Visual Object Recognition
The Problem: Visual Object Recognition
The Problem: Visual Object Recognition

• Fast
The Problem: Visual Object Recognition

- Fast
- Accurate
The Problem:
Visual Object Recognition

- Fast
- Accurate
- Tolerant to variation
The Problem: Visual Object Recognition

- Fast
- Accurate
- Tolerant to variation
- Effortless
The Problem: Visual Object Recognition

- **Fast**
- **Accurate**
- **Tolerant to variation**
- **Effortless**
- **Critical to survival**
The Problem: Visual Object Recognition

- Fast
- Accurate
- Tolerant to variation
- Effortless
- Critical to survival
  (for primates)
hard?
hard?

// the world is 3D but the retina is 2D
hard?

// the world is 3D but the retina is 2D
// the curse of dimensionality
hard?

the world is 3D but the retina is 2D

the curse of dimensionality

considerable image variation
image variation!

do you recognize me?
image variation!

do you recognize me?
image variation!

do you recognize me?
image variation!

do you recognize me?
image variation!

do you recognize me?
~50% of that is for vision!
you learned it...
Need for speed

Hardware

Software

Science
The Approach:
Reverse Engineering the Brain

REVERSE

Study
Natural System
The Approach: Reverse Engineering the Brain

REVERSE

Study
Natural System

FORWARD

Build
Artificial System
The Approach:
Reverse Engineering the Brain

REVERSE
Study Natural System

FORWARD
Build Artificial System
Reverse Engineering ...

What is this doing?
How is that representation constructed???
The Ventral Visual Stream

How is that representation constructed???
The Ventral Visual Stream

How is that representation constructed??
IT Cortex can do object recognition

Visual Cortex

brain = 20 petaflops!
The need for speed
The need for speed

- billions of neurons and synapses
The need for speed

- **billions** of neurons and synapses

- **large-scale** natural evolution ("high-throughput screening" of neural architectures)
The need for speed

- **billions** of neurons and synapses

- **large-scale** natural evolution ("high-throughput screening" of neural architectures)

- **hours** of unsupervised learning experience
The need for speed

- **billions** of neurons and synapses
- **large-scale** natural evolution ("high-throughput screening" of neural architectures)
- **hours** of unsupervised learning experience
- faithful reproduction of other models (i.e. blend **many highly tuned** techniques)
Wanna Play with The Big Guys?
But it’s too expensive !!!
Take a chance.
Our strategy

Capitalizing on non-scientific high-tech markets and their $billions of R&D...
Our strategy

Capitalizing on non-scientific high-tech markets and their $billions of R&D...

– **Gaming:** GPUs, PlayStation 3 (CellBE)
Capitalizing on non-scientific high-tech markets and their $billions of R&D...

- **Gaming:** GPUs, PlayStation 3 (CellBE)
- **Web 2.0:** Cloud Computing (Amazon, Google)
GPUs (since 2006)

7800 GTX (2006)
OpenGL/Cg
C++/Python

Monster 16GPU (2008)
CUDA
Python

Tesla Cluster (2009)
CUDA/OpenCL
Python
Our 16-GPU Monster-Class Supercomputer

the world’s most compact (18''x18''x18'') and inexpensive ($3000) supercomputer.
Cell Broadband Engine (since 2007)

Teraflop Playstation3 clusters:

DiCarlo Lab / MIT

Cox Lab / Harvard
Amazon Cloud Computing (since 2008)

- Elastic Compute Cloud (EC2)
  - Machine Images
  - On-Demand Instances
- Simple Storage Service (S3)
- SimpleDB
- CloudFront
- Simple Queue Service (SQS)

Integration
Some numbers...

- Performance (gflops)
- Development Time (hours)
Some numbers...

- Performance (gflops)
- Development Time (hours)

<table>
<thead>
<tr>
<th>Platform</th>
<th>Performance</th>
<th>Development Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>C/SSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GT200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thursday, October 1, 2009
Some numbers...

- **Performance (gflops)**
  - Matlab: 0.3
  - C/SSE: 9.0
  - PS3: 9.0
  - GT200: 0.3

- **Development Time (hours)**
  - C/SSE: 9.0

3D Filterbank Convolution

Thursday, October 1, 2009
Some numbers...

- **Matlab**
  - Performance (gflops): 0.3
  - Development Time (hours): 9.0

- **C/SSE**
  - Performance (gflops): 9.0
  - Development Time (hours): 9.0

- **PS3**
  - Performance (gflops): 111.4
  - Development Time (hours): 111.4

- **GT200**
  - Performance (gflops): 111.4
  - Development Time (hours): 111.4

*3D Filterbank Convolution*
Some numbers...

- **Matlab**
  - Performance (gflops): 0.3
  - Development Time (hours): 3

- **C/SSE**
  - Performance (gflops): 9.0
  - Development Time (hours): 9

- **PS3**
  - Performance (gflops): 111.4

- **GT200**
  - Performance (gflops): 339.3
  - Development Time (hours): 3
Some numbers...

---

**Performance (gflops)**
- Matlab: 0.3
- C/SSE: 9.0
- PS3: 111.4
- GT200: 339.3

**Development Time (hours)**
- Matlab: 0.5
- C/SSE: 10.0
- PS3: 111.4
- GT200: 339.3

---

Thursday, October 1, 2009
Some numbers...

<table>
<thead>
<tr>
<th></th>
<th>Performance (gflops)</th>
<th>Development Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>C/SSE</td>
<td>9.0</td>
<td>10.0</td>
</tr>
<tr>
<td>PS3</td>
<td></td>
<td>111.4</td>
</tr>
<tr>
<td>GT200</td>
<td></td>
<td>339.3</td>
</tr>
</tbody>
</table>

Thursday, October 1, 2009
Some numbers...

- **Matlab**
  - Performance (gflops): 0.3
  - Development Time (hours): 0.5

- **C/SSE**
  - Performance (gflops): 9.0
  - Development Time (hours): 10.0

- **PS3**
  - Performance (gflops): 30.0
  - Development Time (hours): 30.0

- **GT200**
  - Performance (gflops): 111.4
  - Development Time (hours): 339.3

---

Thursday, October 1, 2009
Some numbers...

- Q9450 (Matlab) [2008]: 0.3
- Q9450 (C/SSE) [2008]: 9.0
- 7900GTX (Cg) [2006]: 68.2
- PS3/Cell (C/ASM) [2007]: 111.4
- 8800GTX (CUDA) [2007]: 192.7
- GTX280 (CUDA) [2008]: 339.3

Performance (gflops)

3D Filterbank Convolution
Need for speed
Hardware
Software
Science
What do we all want?

- Ease of use
- Maximum raw speed
- Ease of extension
- Hardware “agnostic”
A little story

You just finished your code...
A little story

You just finished your code...

1. You run it on one image: it works!
A little story

You just finished your code...

1. You run it on one image: it works!
2. You adjust your parameters: it’s slow!
A little story

You just finished your code...

1. You run it on one image: it works!
2. You adjust your parameters: it’s slow!
3. Your optimize your code: it’s fast now!
A little story

You just finished your code...

1. You run it on one image: it works!
2. You adjust your parameters: it’s slow!
3. You optimize your code: it’s fast now!
4. You run it on another image: it’s slow now!
A little story

You just finished your code...

1. You run it on one image: it works!
2. You adjust your parameters: it’s slow!
3. Your optimize your code: it’s fast now!
4. You run it on another image: it’s slow now!
5. You repeat or you stop...
Here are the keys to Easy-High-Performance!
Meta-programming?
Meta-programming!

Leave the **grunt-programming** to the computer (i.e. auto-tuning like ATLAS or FFTW)

- Dynamically compile **specialized versions** of the same kernel for different conditions (~Just-in-Time Compilation (JIT))
- **Smooth** syntactic ugliness: unroll loops, index un-indexable registers
- **Dynamic**, empirical run-time **tuning**
Meta-programming!

“Instrumentalize” your solutions:

- Block size
- Work size
- Loop unrolling
- Pre-fetching
- Spilling
- etc.
Meta-programming!

Let the computer find the **optimal code**:
- brute-force search with a **global objective**
- machine-learning approach with **local objectives** and **hidden variables** (advanced)
- eg. PyCuda makes this easy:
  - Access properties of compiled code:
    ```python
    func.{registers,lmem,smem}
    ```
  - Exact GPU timing via events
  - Can calculate hardware-dependent MP occupancy
Meta-programming!

- GPU Metaprogramming using **PyCUDA**: Methods & Applications
  - Andreas Kloeckner (Brown)
  - Friday 1pm @ Empire
How ?
Our mantra: always use the right tool!
How? Python!

You're flying! How?

I dunno... dynamic typing? Whitespace?

Come join us! Programming is fun again! It's a whole new world up here!

But how are you flying?

I just typed import antigravity

That's it?

...I also sampled everything in the medicine cabinet for comparison.

But I think this is the Python.
Meta-programming requires careful engineering.
Meta-programming requires careful engineering
The Approach:
Forward Engineering the Brain

REVERSE
Study
Natural System

FORWARD
Build
Artificial System

Thursday, October 1, 2009
The Approach:
Forward Engineering the Brain

REVERSE
Study Natural System

FORWARD
Build Artificial System
How are things done normally?
How are things done normally?

Usual Formula:
How are things done normally?

Usual Formula:

1) One grad student
How are things done normally?

Usual Formula:

1) One grad student
2) One Model (size limited by runtime)
How are things done normally?

**Usual Formula:**

1) One grad student

2) One Model *(size limited by runtime)*

3) Performance numbers on a few standard test sets
How are things done normally?

**Usual Formula:**

1) One grad student
2) One Model *(size limited by runtime)*
3) Performance numbers on a few standard test sets
4) yay. we. rock.
How are things done normally?

Usual Formula:

1) One grad student
2) One Model (size limited by runtime)
3) Performance numbers on a few standard test sets
4) yay. we. rock.
5) One Ph.D.
How are things done normally?

Usual Formula:

1) One grad student
2) One Model (size limited by runtime)
3) Performance numbers on a few standard test sets
4) yay. we. rock.
5) One Ph.D.
Doing things a little bit differently
1) One grad student
Doing things a little bit differently

1) One grad student

2) One Hundreds of Thousands of BIG Models
Doing things a little bit differently

1) One grad student

2) One Hundreds of Thousands of BIG Models

3) Performance numbers on a few standard test sets
Doing things a little bit differently

1) One grad student

2) One Hundreds of Thousands of BIG Models

3) Performance numbers on a few standard test sets
Doing things a little bit differently

1) One grad student

2) One Hundreds of Thousands of BIG Models

3) Performance numbers on a few standard test sets

4) yay. we. rock.
Doing things a little bit differently

1) One grad student

2) One Hundreds of Thousands of BIG Models

3) Performance numbers on a few standard test sets

4) yay. we. rock.

5) Hundreds of Thousands One PhD?
High-Throughput Screening
Pipeline: Biology

“Plate” a Diversity of Organisms → Allow them to grow → Apply Challenge

Collect Surviving Colonies → Study / Repeat
Pipeline: Biology–Inspired Vision

Generate Random Models → Unsupervised Learning (Video) → Test with “screening” task

Skim off best models → Validate on other tasks
A Broad Parametric Model

Normalize
\[ N_i = \frac{\text{Input}_i}{\text{norm}(\text{Input}_{\text{neighborhd}})} \]

Compute Filter Responses
\[ R_i = F_i \otimes N \]
\[ R_i < \text{thresh}: R_i = \text{thresh} \]
\[ R_i > \text{sat}: R_i = \text{sat} \]

Determine a “Winning Filter”
\[ R_i' = (\sum T_k \ast H_k) \ast R_i \]
\[ \text{winner: max}(R_i') \]

Update Filter
\[ F_{\text{winning}} = F_{\text{winning}} + \text{learning rate} \ast N \]
A Broad Parametric Model

Normalize
\[ N_i = \frac{\text{Input}_i}{\text{norm(\text{Input}_{\text{neighborhd}})}} \]

Compute Filter Responses
\[ R_i = F_i \otimes N \]
\[ R_i < \text{thresh}: R_i = \text{thresh} \]
\[ R_i > \text{sat}: R_i = \text{sat} \]

Determine a “Winning Filter”
\[ R_i' = (\sum T_k * H_k) * R_i \]
winner: max(R_i')

Update Filter
\[ F_{\text{winning}} = F_{\text{winning}} + \text{learning rate} * N \]

• Optimize “Coverage”
A Broad Parametric Model

Normalize
\[ N_i = \frac{\text{Input}_i}{\text{norm(\text{Input}_{\text{neighborhood}})}} \]

Compute Filter Responses
\[ R_i = F_i \otimes N \]
R_i < thresh: \( R_i = \text{thresh} \)
R_i > sat: \( R_i = \text{sat} \)

Determine a “Winning Filter”
\[ R'_i = (\Sigma T_k * H_k) * R_i \]
winner: max(\( R'_i \))

Update Filter
\[ F_{\text{winning}} = F_{\text{winning}} + \text{learning rate} \times N \]

• Optimize “Coverage”
  (filters span the range of observed inputs)
A Broad Parametric Model

Normalize
\[ N_i = \text{Input}_i / \text{norm(Input}_{\text{neighborhd}}) \]

Compute Filter Responses
\[ R_i = F_i \otimes N \]
\[ R_i < \text{thresh}: R_i = \text{thresh} \]
\[ R_i > \text{sat}: R_i = \text{sat} \]

Determine a “Winning Filter”
\[ R'_i = (\Sigma T_k * H_k) * R_i \]
winner: max(R'_i)

Update Filter
\[ F_{\text{winning}} = F_{\text{winning}} + \text{learning rate} * N \]

• Optimize “Coverage” (filters span the range of observed inputs)

• Privilege movement of filters in certain directions using temporal information
A Broad Parametric Model

Normalize
\[ N_i = \frac{\text{Input}_i}{\text{norm}(\text{Input}_{\text{neighborhd}})} \]

Compute Filter Responses
\[ R_i = F_i \otimes N \]
\[ R_i < \text{thresh}: R_i = \text{thresh} \]
\[ R_i > \text{sat}: R_i = \text{sat} \]

Determine a “Winning Filter”
\[ R'_i = (\sum T_k * H_k) * R_i \]
winner: \[ \max(R'_i) \]

Update Filter
\[ F_{\text{winning}} = F_{\text{winning}} + \text{learning rate} * N \]

- Optimize “Coverage” (filters span the range of observed inputs)
- Privilege movement of filters in certain directions using temporal information
- Expand dimensionality greatly and then scale back as layers progress
State-of-the-art performance

**d. MultiPIE Hybrid**

- regular + v1-like (control)
- SIFT GB PHOG PHOW SLF state-of-the-art (from literature)
- 2691 1261 2005 2312 3281 blend top 5 models (high-throughput search)

Pinto, DiCarlo, Cox (in review)
State-of-the-art performance

a. Cars vs. Planes (validation)

b. Boats vs. Animals

c. Synthetic Faces

d. MultiPIE Hybrid

Pinto, DiCarlo, Cox (in review)
State-of-the-art performance

Pinto, Cox, DiCarlo PLoS08
State-of-the-art performance

Pinto, DiCarlo, Cox ECCV08

ORL Face Set

8 training examples

Performance (% correct)

pixel

VI-like

perfect!

chance

(1/40=2.5%)
State-of-the-art performance

Pinto, DiCarlo, Cox ECCV08
State-of-the-art performance

<table>
<thead>
<tr>
<th>Reference</th>
<th>Methods</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huang08 [6]</td>
<td>Nowak [8]</td>
<td>73.93%±0.49</td>
</tr>
<tr>
<td></td>
<td>MERL</td>
<td>70.52%±0.60</td>
</tr>
<tr>
<td></td>
<td>Nowak+MERL</td>
<td>76.18%±0.58</td>
</tr>
<tr>
<td>Wolf08 [17]</td>
<td>descriptor-based</td>
<td>70.62%±0.57</td>
</tr>
<tr>
<td></td>
<td>one-shot-learning*</td>
<td>76.53%±0.54</td>
</tr>
<tr>
<td></td>
<td>hybrid*</td>
<td>78.47%±0.51</td>
</tr>
<tr>
<td>This paper</td>
<td>Pixels/MKL</td>
<td>68.22%±0.41</td>
</tr>
<tr>
<td></td>
<td>V1-like/MKL</td>
<td>79.35%±0.55</td>
</tr>
</tbody>
</table>

Table 3. Average performance comparison with the current state-of-the-art on LFW. *note that the “one-shot-learning” and “hybrid” methods from [17] can’t directly be compared to ours as they exploit the fact that individuals in the training and testing sets are mutually exclusive (i.e. using this property, you can build a powerful one-shot-learning classifier knowing that each test example is different from all the training examples, see [17] for more details. Our decision not to use such techniques effectively handicaps our results relative to reports that use them).

Figure 1. ROC curve comparison with the current state-of-the-art on LFW. These curves were generated using the standard procedure described in [24].

Pinto, DiCarlo, Cox CVPR09
Acknowledgements
COME

"Thank You"
AGAIN
and bring
a Friend!

Thursday, October 1, 2009
Back Pocket Slides

slide by David Cox