Efficient Computing for Robotics and AI

Vivienne Sze
Massachusetts Institute of Technology

In collaboration with Luca Carlone, Yu-Hsin Chen, Joel Emer, Sertac Karaman, Tushar Krishna, Trevor Henderson, Peter Li, Fangchang Ma, James Noraky, Amr Suleiman, Diana Wofk, Tien-Ju Yang, Zhengdong Zhang

Contact Info
email: sze@mit.edu
website: www.rle.mit.edu/eems
Processing at “Edge” instead of the “Cloud”

Communication  Privacy  Latency
Cameras and radar generate ~6 gigabytes of data every 30 seconds.

Self-driving car prototypes use approximately 2,500 Watts of computing power.

Generates wasted heat and some prototypes need water-cooling!
Robots Consuming < 1 Watt for Actuation

Low Energy Robotics

- Miniature aerial vehicles
- Lighter than air vehicles
- Micro unmanned gliders
- Miniature satellites
Existing Processors Consume Too Much Power

< 1 Watt

> 10 Watts
Transistors are NOT Getting More Efficient

Slow down of Moore’s Law and Dennard Scaling
General purpose microprocessors not getting faster or more efficient

- Need **specialized hardware** for significant improvement in speed and energy efficiency
- Redesign computing hardware from the ground up!
Energy-Efficient Computing with Cross-Layer Design

**Algorithms**
- Convolutions
- Pooling
- Convs
- Linear Classifier
- Object Categories / Positions

**Systems**
- Image processing system

**Architectures**
- Link Clock, Core Clock
- DCNN Accelerator
- 14×12 PE Array

**Circuits**
- On-Chip Buffer
- Spatial PE Array
- Off-Chip DRAM

MIT
## Power Dominated by Data Movement

<table>
<thead>
<tr>
<th>Operation:</th>
<th>Energy (pJ)</th>
<th>Relative Energy Cost</th>
<th>Area (µm²)</th>
<th>Relative Area Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>0.03</td>
<td>1</td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>16b Add</td>
<td>0.05</td>
<td>10</td>
<td>67</td>
<td>10</td>
</tr>
<tr>
<td>32b Add</td>
<td>0.1</td>
<td>100</td>
<td>137</td>
<td>100</td>
</tr>
<tr>
<td>16b FP Add</td>
<td>0.4</td>
<td>1000</td>
<td>1360</td>
<td>1000</td>
</tr>
<tr>
<td>32b FP Add</td>
<td>0.9</td>
<td>10000</td>
<td>4184</td>
<td>10000</td>
</tr>
<tr>
<td>8b Mult</td>
<td>0.2</td>
<td>100000</td>
<td>282</td>
<td>100000</td>
</tr>
<tr>
<td>32b Mult</td>
<td>3.1</td>
<td>1000000</td>
<td>3495</td>
<td>1000000</td>
</tr>
<tr>
<td>16b FP Mult</td>
<td>1.1</td>
<td>10000000</td>
<td>1640</td>
<td>10000000</td>
</tr>
<tr>
<td>32b FP Mult</td>
<td>3.7</td>
<td>100000000</td>
<td>7700</td>
<td>100000000</td>
</tr>
<tr>
<td>32b SRAM Read (8KB)</td>
<td>5</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Memory access is **orders of magnitude** higher energy than compute.

[Horowitz, ISSCC 2014]
Autonomous Navigation Uses a Lot of Data

- **Semantic Understanding**
  - High frame rate
  - Large resolutions
  - Data expansion

- **Geometric Understanding**
  - Growing map size

[Pire, RAS 2017]
Visual-Inertial Localization

Determines location/orientation of robot from images and IMU (also used by headset in Augmented Reality and Virtual Reality)

*Subset of SLAM algorithm (Simultaneous Localization And Mapping)
Localization at under 25 mW

*First chip* that performs **complete** Visual-Inertial Odometry

**Front-End for camera**
*(Feature detection, tracking, and outlier elimination)*

**Front-End for IMU**
*(pre-integration of accelerometer and gyroscope data)*

**Back-End Optimization of Pose Graph**

Consumes *684×* and *1582×* less energy than mobile and desktop CPUs, respectively

*Joint work with Sertac Karaman (AeroAstro)*

[Zhang, RSS 2017], [Suleiman, VLSI 2018]
Key Methods to Reduce Data Size

**Navion:** Fully integrated system – no off-chip processing or storage

![Flowchart diagram showing data flow and methods](image)

- **Apply Low Cost Frame Compression**
- **Exploit Sparsity in Graph and Linear Solver**

Use *compression* and *exploit sparsity* to reduce memory down to 854kB

[Suleiman et al., VLSI 2018]
Frame Buffer Memory

Compress

4x4 pixels example

Line Buffer (1.4 kB)

5-bit pixels

Find Min. & Max.

Max.

Min.

Thresh

\( \geq? \)

\( \gg 1 \)

000010001111000

16 bits

Frame Memory (78 kB)

Decompress

5 bits

5 bits

16 bits

0000010001111000

4444

4744

4777

7444

Original Image (352.5 kB)

8-bit/pixel

4.4x reduction

Compressed Image (79.4 kB)

1.625-bit/pixel

[Suleiman et al., VLSI 2018]
Linear Solver: \( H\Delta x = \varepsilon \), solve for \( \Delta x \)

1. \( H = LL^T \)  
   Calculate \( L \)

2. \( Lu = \varepsilon \)  
   Solve for \( u \)

3. \( L^T \Delta x = u \)  
   Solve for \( \Delta x \)

Linear Solver and Hessian Memory

Sparsity pattern in both \( H \) & \( L \)  
(Non-zero: black)

Memory size
- Full: 703 kB
- Sym: 353 kB
- Sym + Sparse: 134 kB

Processing time
- Full: 48.2 ms
- Sparse: 6.7 ms

Non-zero Hessian (134 kB)

Sparse-based Control

Physical Address

Masked Read/Write

Non-zero entry

Zero

Linearize

Input
- Row
- Column
- Read/Write

Output

Linearize

Back-substitute

Cholesky

Matrix

\( H \) & \( \varepsilon \)
Factor Graph Memory

- **One Memory** (962 kB)
  - 20 KFs * 200 features/KF * 10 feature age = 40,000
  - 5.4x
- **Two-stage Memory** (177 kB)
  - 40,000

[Suleiman et al., VLSI 2018]
Navion Evaluation

• **Peak Performance @ Maximum Configuration**
  - VFE: 28 – 171 fps (71 fps average)
  - BE: 16 – 90 fps (19 fps average)
  - Average Power Consumption: 24mW
  - Trajectory Error: 0.28%

• **Real-Time Performance @ Optimized Configuration**
  - VF: 20 fps
  - BE: 5 fps
  - Average Power Consumption: 2mW
  - Trajectory Error: 0.27%

*Over 250 configurable parameters to adapt to different sensors and environments*

[http://navion.mit.edu](http://navion.mit.edu)

Evaluated on EuRoC dataset

[Suleiman et al., VLSI 2018]
State-of-the-art approaches use Deep Neural Networks, which require up to several hundred millions of operations and weights to compute!

>100x more complex than video compression
Deep Neural Networks (DNNs) have become a cornerstone of AI.
Properties We Can Leverage

- Operations exhibit **high parallelism** → **high throughput** possible
- Memory Access is the Bottleneck

**Memory Read**

- DRAM to ALU: filter weight, image pixel, partial sum

**MAC**

- ALU: multiply-and-accumulate

**Memory Write**

- ALU to DRAM: updated partial sum

**Worst Case**: all memory R/W are **DRAM** accesses

- Example: AlexNet has **724M** MACs → **2896M** DRAM accesses required
Properties We Can Leverage

- Operations exhibit **high parallelism** → **high throughput** possible
- **Input data reuse** opportunities (**up to 500x**)
Exploit Data Reuse at Low-Cost Memories

Specialized hardware with small (<1kB) low cost memory near compute

Normalized Energy Cost*

1x (Reference)

2x

6x

200x

* measured from a commercial 65nm process

Farther and larger memories consume more power
Weight Stationary (WS)

- Minimize weight read energy consumption
  - maximize convolutional and filter reuse of weights

- Examples:
  - [Chakradhar, ISCA 2010]
  - [nn-X (NeuFlow), CVPRW 2014]
  - [Park, ISSCC 2015]
  - [Origami, GLSVLSI 2015]
Output Stationary (OS)

- Minimize partial sum R/W energy consumption
  - maximize local accumulation

- Examples:
  [Gupta, ICML 2015]  [ShiDianNao, ISCA 2015]
  [Peemen, ICCD 2013]
Row Stationary Dataflow

- Maximize row convolutional reuse in RF
  - Keep a filter row and fmap sliding window in RF
- Maximize row psum accumulation in RF
Optimize for overall energy efficiency instead for only a certain data type
**Dataflow Comparison: CONV Layers**

RS optimizes for the best **overall** energy efficiency

[Chen et al., ISCA 2016]
Deep Neural Networks at Under 0.3W

Exploits data reuse for $100x$ reduction in memory accesses from global buffer and $1400x$ reduction in memory accesses from off-chip DRAM

Overall $>10x$ energy reduction compared to a mobile GPU (Nvidia TK1)

Results for AlexNet

[Joint work with Joel Emer]  http://eyeriss.mit.edu
Features: Energy vs. Accuracy

Exponential

Energy/ Pixel (nJ)

Accuracy (Average Precision)

Measured in on VOC 2007 Dataset
1. DPM v5 [Girshick, 2012]

* Only feature extraction. Does not include data, classification energy, augmentation and ensemble, etc.

Measured in 65nm*

[Suleiman, VLSI 2016]  [Chen, ISSCC 2016]
Energy-Efficient Processing of DNNs

A significant amount of algorithm and hardware research on energy-efficient processing of DNNs

Hardware Architectures for Deep Neural Networks

ISCA Tutorial

June 24, 2017

Website: http://eyeriss.mit.edu/tutorial.html

http://eyeriss.mit.edu/tutorial.html

V. Sze, Y.-H. Chen, T-J. Yang, J. Emer,

We identified various limitations to existing approaches
Design of Efficient DNN Algorithms

- Popular efficient DNN algorithm approaches

**Network Pruning**

Before pruning

After pruning

**Compact Network Architectures**

Examples: SqueezeNet, MobileNet

... also reduced precision

- Focus on reducing **number of MACs and weights**
- Does it translate to energy savings?
Data Movement is Expensive

Energy of weight depends on memory hierarchy and dataflow

Normalized Energy Cost*

1× (Reference)

1×

2×

6×

200×

* measured from a commercial 65nm process
Energy-Evaluation Methodology

DNN Shape Configuration
(# of channels, # of filters, etc.)

DNN Weights and Input Data
[0.3, 0, -0.4, 0.7, 0, 0, 0.1, …]

Hardware Energy Costs of each MAC and Memory Access

Memory Accesses Optimization

# of MACs Calculation

# acc. at mem. level 1
# acc. at mem. level 2
⋯
# acc. at mem. level n

# of MACs

Energy

Energy Consumption

Tool available at: https://energyestimation.mit.edu/

[Yang et al., CVPR 2017]
Key Observations

- Number of weights *alone* is not a good metric for energy
- *All data types* should be considered

Energy Consumption of GoogLeNet

- **Output Feature Map**: 43%
- **Input Feature Map**: 25%
- **Weights**: 22%
- **Computation**: 10%

[Yang et al., CVPR 2017]
Directly target energy and incorporate it into the optimization of DNNs to provide greater energy savings

- Sort layers based on energy and prune layers that consume most energy first
- EAP reduces AlexNet energy by **3.7x** and outperforms the previous work that uses magnitude-based pruning by **1.7x**

Pruned models available at [http://eyeriss.mit.edu/energy.html](http://eyeriss.mit.edu/energy.html)

[Yang et al., CVPR 2017]
NetAdapt: Platform-Aware DNN Adaptation

- **Automatically adapt DNN** to a mobile platform to reach a target latency or energy budget
- **Use empirical measurements** to guide optimization (avoid modeling of tool chain or platform architecture)

---

NetAdapt

**Budget**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>3.8</td>
</tr>
<tr>
<td>Energy</td>
<td>10.5</td>
</tr>
</tbody>
</table>

**Empirical Measurements**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Proposal A</th>
<th>...</th>
<th>Proposal Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>15.6</td>
<td>...</td>
<td>14.3</td>
</tr>
<tr>
<td>Energy</td>
<td>41</td>
<td>...</td>
<td>46</td>
</tr>
</tbody>
</table>

**Pretrained Network**

**Adapted Network**

**Network Proposals**

A  B  C  D  Z

[Yang et al., ECCV 2018]

In collaboration with Google’s Mobile Vision Team
Improved Latency vs. Accuracy Tradeoff

- NetAdapt boosts the real inference speed of MobileNet by up to 1.7x with higher accuracy.

*Tested on the ImageNet dataset and a Google Pixel 1 CPU

Reference:

[Yang et al., ECCV 2018]
FastDepth: Fast Monocular Depth Estimation

Depth estimation from a single RGB image desirable, due to the relatively low cost and size of monocular cameras.

**Auto Encoder DNN Architecture (Dense Output)**

- **Encoding Layers**
  - MobileNet
  - $7 \times 7 \times 1024$

- **Decoding Layers**
  - Upsample layers 1 to 5
  - $14 \times 14 \times 512$
  - $28 \times 28 \times 256$
  - $56 \times 56 \times 128$
  - $112 \times 112 \times 64$
  - $224 \times 224 \times 32$

- **Dense Depth Map**
  - $224 \times 224 \times 1$

**Reduction** (similar to classification)

**Expansion**
FastDepth: Fast Monocular Depth Estimation

Apply NetAdapt, compact network design, and depth wise decomposition to decoder layer to enable depth estimation at high frame rates on an embedded platform while still maintaining accuracy.

Configuration: Batch size of one (32-bit float)

Models available at http://fastdepth.mit.edu

[Wofk*, Ma* et al., ICRA 2019]
Many Efficient DNN Design Approaches

Network Pruning

before pruning

after pruning

pruning synapses

pruning neurons

Compact Network Architectures

Convolutional Layer

Depth-Wise Layer

Point-Wise Layer

Reduce Precision

32-bit float 101001010000000000001010000000000100

8-bit fixed 01100110

Binary 0

No guarantee that DNN algorithm designer will use a given approach.

Need flexible hardware!

[Chen et al., SysML 2018]
Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

• Wide range of filter shapes
  – Large and Compact

• Different Layers
  – CONV, FC, depth wise, etc.

• Wide range of sparsity
  – Dense and Sparse

• Scalable architecture

Over an order of magnitude faster and more energy efficient than Eyeriss v1

[Chen et al., JETCAS 2019]

Speed up over Eyeriss v1 scales with number of PEs

<table>
<thead>
<tr>
<th># of PEs</th>
<th>256</th>
<th>1024</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>17.9x</td>
<td>71.5x</td>
<td>1086.7x</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>10.4x</td>
<td>37.8x</td>
<td>448.8x</td>
</tr>
<tr>
<td>MobileNet</td>
<td>15.7x</td>
<td>57.9x</td>
<td>873.0x</td>
</tr>
</tbody>
</table>

[Joint work with Joel Emer]
**Robot Exploration:** Decide where to go by computing Shannon Mutual Information

Where to scan? \[ \begin{array}{c} \text{Select candidate scan locations} \end{array} \] \[ \begin{array}{c} \text{Compute Shannon MI and choose best location} \end{array} \] \[ \begin{array}{c} \text{Move to location and scan} \end{array} \] \[ \begin{array}{c} \text{Update Occupancy Map} \end{array} \]

Mutual Information

Updated Map

[Joint work with Sertac Karaman]
Information Theoretic Mapping

Occupancy grid map, $M$

Mutual information map, $I(M; Z)$

\[
H(M | Z) = H(M) - I(M; Z)
\]

Perspective updated map entropy

Current map entropy

Mutual information
FSMI: Fast Shannon Mutual Information

Shannon Mutual Information
(between beam Z and map M)

\[ I(M; Z) = \sum_{i=1}^{n} \int_{z \geq 0} P(z) f(\delta_i(z), r_i) \, dz \]

No closed form solution. Requires expensive numerical integration at resolution \( \lambda_z \). \( \mathcal{O}(n^2 \lambda_z) \)

FSMI: Fast Shannon Mutual Information

Evaluates MI for all cells in entire beam altogether removes numerical integration. \( \mathcal{O}(n^2) \)

Approximate FSMI

Approximate noise model of depth sensor with truncated Gaussian*. \( \mathcal{O}(n) \)

\*Charrow et al., ICRA 2015

\[ I(M; Z) = \sum_{j=1}^{n} \sum_{k=1}^{n} P(e_j) C_k G_{k,j} \]

\[ I(M; Z) = \sum_{j=1}^{n} \sum_{k=j-\Delta}^{j+\Delta} P(e_j) C_k G_{k,j} \]
FSMI: Fast Shannon Mutual Information

<table>
<thead>
<tr>
<th>Original MI(^1)</th>
<th>FSMI</th>
<th>CSQMI(^2)</th>
<th>Approximate FSMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(n^2 \lambda_z))</td>
<td>(O(n^2))</td>
<td>(O(n))</td>
<td>(O(n))</td>
</tr>
</tbody>
</table>


Measured run time per beam (µsec) on an Intel Xeon core (desktop)

```
Original MI  | 188046
FSMI         | 132
CSQMI        | 29
Approximate FSMI | 17
```

Measured run time per beam (µsec) on an ARM Cortex-A57 core (embedded)

```
CSQMI        | 422
Approximate FSMI | 149
```

Approximate FSMI is over 1000x faster than original MI and 1.7 – 2.8x faster than CSQMI.
Experimental Results (4x Real Time)

Exploration with a mini race car using motion capture for localization

Occupancy map with planned path using RRT* (compute MI on all possible paths)

MI surface

[Zhang et al., ICRA 2019]
Quality of Result

Experiment Environment

Paths with high MI per meter in green

Complete map and trajectory

Approximate FSMI reduces entropy of map at same rate as CSQMI while computing Shannon Mutual Information

Compute time per beam

CSQMI = 422.7 μsec
Approximate FSMI = 111.4 μsec

[Zhang et al., ICRA 2019]
Building Hardware to Compute MI

**Motivation:** Compute MI faster for faster exploration!

Approximate FSMI

\[
I(M; Z) = \sum_{j=1}^{n} \sum_{k=j-\Delta}^{j+\Delta} P(e_j)C_k G_{k,j}
\]

Algorithm is *embarrassingly* parallel!
High throughput should be possible with multiple cores.

Process beams in parallel with multiple cores
Challenge is Data Delivery to All Cores

Power consumption of memory scales with number of ports. Low power SRAM limited to two-ports!

Data delivery, specifically memory bandwidth, limits the throughput (not compute)
Specialized Memory Architecture

Break up map into separate memory banks and novel storage pattern to minimize read conflicts when processing different beams in parallel.

[Li et al., RSS 2019]
Specialized banking, efficient memory arbiter and packing multiple values at each address results in throughput within 94% of the theoretical limit (unlimited bandwidth)

[Li et al., RSS 2019]
Experimental Results

Each FSMI core on FPGA faster than CPU core by 10x

<table>
<thead>
<tr>
<th>Number of cells in a beam</th>
<th>Computation time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.29</td>
</tr>
<tr>
<td>50</td>
<td>0.93</td>
</tr>
<tr>
<td>100</td>
<td>1.7</td>
</tr>
<tr>
<td>200</td>
<td>3.3</td>
</tr>
</tbody>
</table>

Compute FSMI on more locations reduce trajectory length

- 25x FSMI
- 1x FSMI

Map entropy vs. Trajectory length

Compute the mutual information for an entire map of 20m x 20m at 0.1m resolution in under a second while consuming under 2W on an FPGA*

*estimate another order of magnitude reduction with ASIC

[Li et al., RSS 2019]
Extend FSMI to 3D Environments

Computing MI on a 3D map requires significant amounts of storage and compute.

Compress map with OctoMap
[Hornung, et al., Autonomous Robots, 2013]
The 1D occupancy vector consists of multiple segments of repeated occupancy values.
FSMI on Compressed Input

Uncompressed input format

\[ o_1, o_1, ..., o_1, o_2, ..., o_2, o_3, ..., o_3, ..., o_{n_r}, ..., o_{n_r} \]

\[ L_1 + L_2 + L_3 + L_{n_r} = n \]

Compressed format (Run Length Encoding)

\[ (o_1, L_1), (o_2, L_2), (o_3, L_3), ..., (o_{n_r}, L_{n_r}) \]

\[ n_r \]

Time complexity of Approx FSMI

\[ O(n) \]

Goal: achieve the complexity of

\[ O(n_r) \]

\[ n_r \ll n, \text{ significant reduction if the constants are comparable} \]
## Complexity of 2D and 3D FSMI

<table>
<thead>
<tr>
<th></th>
<th>FSMI</th>
<th>Approximate FSMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>$O(n^2)$</td>
<td>$O(n\Delta)$</td>
</tr>
<tr>
<td>3D (compress with RLE)</td>
<td>$O(n_r^2)$</td>
<td>$O(n_r\Delta)$</td>
</tr>
</tbody>
</table>

Measured speed up for a beam of 256 cells on an Intel Xeon CPU core for different degrees of compression (L)

<table>
<thead>
<tr>
<th>Approx FSMI-RLE</th>
<th>$L = 1$</th>
<th>$L = 2$</th>
<th>$L = 4$</th>
<th>$L = 8$</th>
<th>$L = 16$</th>
<th>$L = 32$</th>
<th>$L = 64$</th>
<th>$L = 128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleration</td>
<td>0.2×</td>
<td>0.7×</td>
<td>1.8×</td>
<td>4.6×</td>
<td>7.4×</td>
<td>11.2×</td>
<td>16.5×</td>
<td>24.4×</td>
</tr>
</tbody>
</table>

Baseline (Approx FSMI): 56µsec

Z. Zhang et al., FSMI: Fast computation of Shannon Mutual Information for information-theoretic mapping, *arXiv 2019*

Experiments of 3D FSMI (4x Real Time)

[Z. Zhang et al., arXiv 2019]
Experiments of 3D FSMI

We achieve an average compression ratio of around $18\times$, with an acceleration ratio of $8\times$. 
Low Power 3D Time of Flight Imaging

• Pulsed Time of Flight: Measure distance using round trip time of laser light for each image pixel
  – Illumination + Imager Power: 2.5 – 20 W for range from 1 - 8 m

• Use computer vision techniques and passive images to estimate changes in depth without turning on laser
  – CMOS Imaging Sensor Power: < 350 mW

Real-time Performance on Embedded Processor
VGA @ 30 fps on Cortex-A7 (< 0.5W active power)

[Noraky et al., ICIP 2017]
Results of Low Power Depth ToF Imaging

Mean Relative Error: 0.7%
Duty Cycle (on-time of laser): 11%

[Noraky et al., ICIP 2017]
Summary

• Efficient computing is critical for advancing the progress of autonomous robots, particularly at the smaller scales. → Critical step to making autonomy ubiquitous!

• In order to meet computing demands in terms of power and speed, need to redesign computing hardware from the ground up → Focus on data movement!

• Specialized hardware opens up new opportunities for the co-design of algorithms and hardware → Innovation opportunities for the future of robotics!

Algorithms ↔ Hardware

Today’s slides available at www.rle.mit.edu/eems
Acknowledgements

Research conducted in the MIT Energy-Efficient Multimedia Systems Group would not be possible without the support of the following organizations:

For updates on our research Follow @eems_mit
References

• Energy-Efficient Hardware for Deep Neural Networks
  – Project website: http://eyeriss.mit.edu

• Limitations of Existing Efficient DNN Approaches
  – Hardware Architecture for Deep Neural Networks: http://eyeriss.mit.edu/tutorial.html
References

• Co-Design of Algorithms and Hardware for Deep Neural Networks

• Energy-Efficient Visual Inertial Localization
  – Project website: http://navion.mit.edu
References

• Fast Shannon Mutual Information for Robot Exploration

• Low Power Time of Flight Imaging