Efficient Processing for Deep Learning: Challenges and Opportunities

Vivienne Sze

Massachusetts Institute of Technology

Contact Info
email: sze@mit.edu
website: www.rle.mit.edu/eems

In collaboration with
Yu-Hsin Chen, Joel Emer, Tien-Ju Yang
Video is the Biggest Big Data

Over 70% of today’s Internet traffic is video
Over 300 hours of video uploaded to YouTube every minute
Over 500 million hours of video surveillance collected every day

Energy limited due to battery capacity
Power limited due to heat dissipation

Need energy-efficient pixel processing!
Deep Convolutional Neural Networks

Modern \textit{deep} CNN: up to 1000 CONV layers

CONV Layer $\rightarrow$ Low-level Features $\rightarrow$ CONV Layer $\rightarrow$ High-level Features
Deep Convolutional Neural Networks

CONV Layer → Low-level Features → CONV Layer → High-level Features → FC Layers → Classes

1 – 3 layers
Deep Convolutional Neural Networks

Convolutions account for more than 90% of overall computation, dominating runtime and energy consumption.
High-Dimensional CNN Convolution

Input Image (Feature Map)

Filter

R

R

H

H
High-Dimensional CNN Convolution

Input Image (Feature Map)

Filter

Element-wise Multiplication
High-Dimensional CNN Convolution

Input Image (Feature Map) → Output Image

Filter → Element-wise Multiplication → Partial Sum (psum) Accumulation

a pixel
High-Dimensional CNN Convolution

Input Image (Feature Map)  Output Image

Filter

Sliding Window Processing

a pixel
High-Dimensional CNN Convolution

Input Image

Filter

Output Image

Many Input Channels (C)

AlexNet: 3 – 192 Channels (C)
High-Dimensional CNN Convolution

Many Filters (M)

Input Image

Output Image

Many Output Channels (M)

AlexNet: 96 – 384 Filters (M)
High-Dimensional CNN Convolution

Many Input Images (N)

Filters

Many Output Images (N)

Image batch size: 1 – 256 (N)
### Large Sizes with Varying Shapes

#### AlexNet Convolutional Layer Configurations

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter Size (R)</th>
<th># Filters (M)</th>
<th># Channels (C)</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11x11</td>
<td>96</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5x5</td>
<td>256</td>
<td>48</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3x3</td>
<td>384</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3x3</td>
<td>384</td>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3x3</td>
<td>256</td>
<td>192</td>
<td>1</td>
</tr>
</tbody>
</table>

1. [Krizhevsky, NIPS 2012]
Popular CNNs

• LeNet (1998)
• AlexNet (2012)
• OverFeat (2013)
• VGGNet (2014)
• GoogleNet (2014)
• ResNet (2015)

ImageNet: Large Scale Visual Recognition Challenge (ILSVRC)

Accuracy (Top 5 error)

[O. Russakovsky et al., IJCV 2015]
## Summary of Popular CNNs

<table>
<thead>
<tr>
<th>Metrics</th>
<th>LeNet-5</th>
<th>AlexNet</th>
<th>VGG-16</th>
<th>GoogLeNet (v1)</th>
<th>ResNet-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-5 error</td>
<td>n/a</td>
<td>16.4</td>
<td>7.4</td>
<td>6.7</td>
<td>5.3</td>
</tr>
<tr>
<td>Input Size</td>
<td>28x28</td>
<td>227x227</td>
<td>224x224</td>
<td>224x224</td>
<td>224x224</td>
</tr>
<tr>
<td># of CONV Layers</td>
<td>2</td>
<td>5</td>
<td>16</td>
<td>21 (depth)</td>
<td>49</td>
</tr>
<tr>
<td>Filter Sizes</td>
<td>5</td>
<td>3, 5, 11</td>
<td>3</td>
<td>1, 3, 5, 7</td>
<td>1, 3, 7</td>
</tr>
<tr>
<td># of Channels</td>
<td>1, 6</td>
<td>3 - 256</td>
<td>3 - 512</td>
<td>3 - 1024</td>
<td>3 - 2048</td>
</tr>
<tr>
<td># of Filters</td>
<td>6, 16</td>
<td>96 - 384</td>
<td>64 - 512</td>
<td>64 - 384</td>
<td>64 - 2048</td>
</tr>
<tr>
<td>Stride</td>
<td>1</td>
<td>1, 4</td>
<td>1</td>
<td>1, 2</td>
<td>1, 2</td>
</tr>
<tr>
<td># of Weights</td>
<td>2.6k</td>
<td>2.3M</td>
<td>14.7M</td>
<td>6.0M</td>
<td>23.5M</td>
</tr>
<tr>
<td># of MACs</td>
<td>283k</td>
<td>666M</td>
<td>15.3G</td>
<td>1.43G</td>
<td>3.86G</td>
</tr>
<tr>
<td># of FC layers</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># of Weights</td>
<td>58k</td>
<td>58.6M</td>
<td>124M</td>
<td>1M</td>
<td>2M</td>
</tr>
<tr>
<td># of MACs</td>
<td>58k</td>
<td>58.6M</td>
<td>124M</td>
<td>1M</td>
<td>2M</td>
</tr>
<tr>
<td>Total Weights</td>
<td>60k</td>
<td>61M</td>
<td>138M</td>
<td>7M</td>
<td>25.5M</td>
</tr>
<tr>
<td>Total MACs</td>
<td>341k</td>
<td>724M</td>
<td>15.5G</td>
<td>1.43G</td>
<td>3.9G</td>
</tr>
</tbody>
</table>

CONV Layers increasingly important!
Training vs. Inference

Training (determine weights)

Inference (use weights)

Large Datasets

Weights
Challenges
Key Metrics

• **Accuracy**
  – Evaluate hardware using the appropriate DNN model and dataset

• **Programmability**
  – Support multiple applications
  – Different weights

• **Energy/Power**
  – Energy per operation
  – DRAM Bandwidth

• **Throughput/Latency**
  – GOPS, frame rate, delay

• **Cost**
  – Area (size of memory and # of cores)

[Sze et al., CICC 2017]
Opportunities in Architecture
GPUs and CPUs Targeting Deep Learning


**Knights Mill:** next gen Xeon
Phi “optimized for deep learning”

*Use matrix multiplication libraries on CPUs and GPUs*
Accelerate Matrix Multiplication

• Implementation: **Matrix Multiplication (GEMM)**
  - **CPU**: OpenBLAS, Intel MKL, etc
  - **GPU**: cuBLAS, cuDNN, etc

• Optimized by tiling to storage hierarchy
Map DNN to a Matrix Multiplication

- Convert to matrix mult. using the **Toeplitz Matrix**

Convolution:

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\ast
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\]

Toeplitz Matrix (w/ redundant data)

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
2 & 3 & 5 & 6 \\
4 & 5 & 7 & 8 \\
5 & 6 & 8 & 9 \\
\end{bmatrix}
\times
\begin{bmatrix}
1 & 2 & 4 & 5 \\
2 & 3 & 5 & 6 \\
4 & 5 & 7 & 8 \\
5 & 6 & 8 & 9 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 & 2 & 3 & 4 \\
\end{bmatrix}
\]

Data is repeated

**Goal:** Reduced number of operations to increase throughput
Computation Transformations

- Goal: Bitwise same result, but reduce number of operations
- Focuses mostly on compute
Analogy: Gauss’s Multiplication Algorithm

\[(a + bi)(c + di) = (ac - bd) + (bc + ad)i.\]

4 multiplications + 3 additions

\[k_1 = c \cdot (a + b)\]
\[k_2 = a \cdot (d - c)\]
\[k_3 = b \cdot (c + d)\]
Real part = \[k_1 - k_3\]
Imaginary part = \[k_1 + k_2\].

3 multiplications + 5 additions

Reduce number of multiplications, but increase number of additions
Reduce Operations in Matrix Multiplication

- **Winograd** [Lavin, CVPR 2016]
  - **Pro:** 2.25x speed up for 3x3 filter
  - **Con:** Specialized processing depending on filter size

- **Fast Fourier Transform** [Mathieu, ICLR 2014]
  - **Pro:** Direct convolution $O(N_o^2N_f^2)$ to $O(N_o^2\log_2N_o)$
  - **Con:** Increase storage requirements

- **Strassen** [Cong, ICANN 2014]
  - **Pro:** $O(N^3)$ to $(N^{2.807})$
  - **Con:** Numerical stability
cuDNN: Speed up with Transformations

60x Faster Training in 3 Years

AlexNet training throughput on:

CPU: 1x E5-2680v3 12 Core 2.5GHz, 128GB System Memory, Ubuntu 14.04

M40 bar: 8x M40 GPUs in a node, P100: 8x P100 NVLink-enabled

Source: Nvidia
Specialized Hardware (Accelerators)
Properties We Can Leverage

• Operations exhibit **high parallelism** → **high throughput** possible

• Memory Access is the Bottleneck

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

• Example: AlexNet [NIPS 2012] 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 **low-cost memory**

![Diagram showing convolutional reuse (pixels, weights) and image reuse (pixels), as well as filter reuse (weights).]
Highly-Parallel Compute Paradigms

Temporal Architecture (SIMD/SIMT)

- Memory Hierarchy
  - Register File
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - Control

Spatial Architecture (Dataflow Processing)

- Memory Hierarchy
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
  - ALU
Advantages of Spatial Architecture

Temporal Architecture (SIMD/SIMT)

- Efficient Data Reuse
  - Distributed local storage (RF)

- Inter-PE Communication
  - Sharing among regions of PEs

Spatial Architecture (Dataflow Processing)

Memory Hierarchy

- Processing Element (PE)
  - Reg File
  - Control

0.5 – 1.0 kB
Data Movement is Expensive

Maximize data reuse at lower levels of hierarchy
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]
• Minimize **partial sum** R/W energy consumption
  - maximize local accumulation

• **Examples:**
  
  [Gupta, *ICML* 2015]  
  [ShiDianNao, *ISCA* 2015]  
  [Peemen, *ICCD* 2013]
No Local Reuse (NLR)

• Use a large global buffer as shared storage
  – Reduce DRAM access energy consumption

• Examples:
  [DianNao, ASPLOS 2014]  [DaDianNao, MICRO 2014]
  [Zhang, FPGA 2015]
Row Stationary Dataflow

Row 1

PE 1

Row 2

PE 4

Row 3

PE 7

Row 1

PE 2

Row 2

PE 5

Row 3

PE 8

Row 3

PE 3

Row 2

PE 6

Row 4

PE 9

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1

Row 2

Row 3

Row 1
RS uses 1.4× – 2.5× lower energy than other dataflows

[Chen, ISCA 2016]
Eyeriss Deep CNN Accelerator

[Chen et al., ISSCC 2016]
## Comparison with GPU

<table>
<thead>
<tr>
<th></th>
<th><strong>Eyeriss</strong></th>
<th><strong>NVIDIA TK1 (Jetson Kit)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technology</strong></td>
<td>65nm</td>
<td>28nm</td>
</tr>
<tr>
<td><strong>Clock Rate</strong></td>
<td>200MHz</td>
<td>852MHz</td>
</tr>
<tr>
<td><strong># Multipliers</strong></td>
<td>168</td>
<td>192</td>
</tr>
<tr>
<td><strong>On-Chip Storage</strong></td>
<td>Buffer: 108KB, Spad: 75.3KB</td>
<td>Shared Mem: 64KB, Reg File: 256KB</td>
</tr>
<tr>
<td><strong>Word Bit-Width</strong></td>
<td>16b Fixed</td>
<td>32b Float</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>34.7 fps</td>
<td>68 fps</td>
</tr>
<tr>
<td><strong>Measured Power</strong></td>
<td>278 mW</td>
<td>Idle/Active: 3.7W/10.2W</td>
</tr>
<tr>
<td><strong>DRAM Bandwidth</strong></td>
<td>127 MB/s</td>
<td>1120 MB/s</td>
</tr>
</tbody>
</table>

1. AlexNet Convolutional Layers Only
2. Board Power
3. Modeled from [Tan, SC11]

[http://eyeriss.mit.edu](http://eyeriss.mit.edu)
Features: Energy vs. Accuracy

Measured in 65nm*
1. [Suleiman, VLSI 2016]
2. [Chen, ISSCC 2016]

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

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

[Suleiman et al., ISCAS 2017]
Opportunities in Joint Algorithm Hardware Design
Approaches

• **Reduce size of operands for storage/compute**
  – Floating point $\rightarrow$ Fixed point
  – Bit-width reduction
  – Non-linear quantization

• **Reduce number of operations for storage/compute**
  – Exploit Activation Statistics (Compression)
  – Network Pruning
  – Compact Network Architectures
Commercial Products using 8-bit Integer

Reduced Precision in Research

• Reduce number of bits
  – Binary Nets [Courbariaux, NIPS 2015]

• Reduce number of unique weights
  – Ternary Weight Nets [Li, arXiv 2016]
  – XNOR-Net [Rategari, ECCV 2016]

• Non-Linear Quantization
  – LogNet [Lee, ICASSP 2017]
Sparsity in Feature Maps

Many **zeros** in output fmaps after ReLU

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>-1</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>6</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

ReLU

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Graph showing the number of activations and non-zero activations for CONV layers, normalized.
Exploit Sparsity

Method 1: Skip memory access and computation

Method 2: Compress data to reduce storage and data movement

[Chen et al., ISSCC 2016]
Pruning – Make Weights Sparse

Optimal Brain Damage
[Lecun et al., NIPS 1989]

Prune DNN based on **magnitude** of weights
[Han et al., NIPS 2015]

Example: AlexNet
**Weight Reduction:**
CONV layers 2.7x, **FC layers 9.9x**
**Overall Reduction:**
Weights 9x, MACs 3x
Network Architecture Design

Build Network with series of Small Filters

GoogleNet/Inception v3

5x5 filter

Apply sequentially

decompose

5x1 filter

separable filters

VGG-16

5x5 filter

Apply sequentially

decompose

Two 3x3 filters

48
1x1 Bottleneck in Popular DNN models

ResNet

GoogleNet

SqueezeNet

compress

expand

relu

relu

relu

relu

 Previous layer
Key Metrics for Embedded DNN

- Accuracy $\rightarrow$ Measured on Dataset
- Speed $\rightarrow$ Number of MACs
- Storage Footprint $\rightarrow$ Number of Weights
- Energy $\rightarrow$ ?
Energy-Evaluation Methodology

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

CNN Weights and Input Data

[0.3, 0, -0.4, 0.7, 0, 0, 0.1, ...]

Hardware Energy Costs of each
MAC and Memory Access

Energy estimation tool available at http://eyeriss.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]
Deeper CNNs with fewer weights do not necessarily consume less energy than shallower CNNs with more weights

[Yang et al., CVPR 2017]
Magnitude-based Weight Pruning

Reduce number of weights by removing small magnitude weights
Energy-Aware Pruning

Remove weights from layers in order of highest to lowest energy

3.7x reduction in AlexNet / 1.6x reduction in GoogLeNet

[Yang et al., CVPR 2017]
Summary

• **Energy-Efficient Approaches**
  – Minimize data movement
  – Balance flexibility and energy-efficiency
  – Exploit sparsity with joint algorithm and hardware design

• **Joint algorithm and hardware design** can deliver additional energy savings (directly target energy)

• **Linear increase in accuracy** requires **exponential increase in energy**

**Acknowledgements:** This work is funded by the DARPA YFA grant, MIT Center for Integrated Circuits & Systems, and gifts from Intel, Nvidia and Google.
References

Overview Paper

More info about Eyeriss and Tutorial on DNN Architectures
http://eyeriss.mit.edu

MIT Professional Education Course on “Designing Efficient Deep Learning Systems”
March 26 – 27, 2018 in Mountain View, CA
http://professional-education.mit.edu/deeplearning

For updates Follow @eems_mit
http://mailman.mit.edu/mailman/listinfo/eems-news