DNN Model and Hardware Co-Design

ISCA Tutorial (2017)
Website: http://eyeriss.mit.edu/tutorial.html

Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang
 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
## Cost of Operations

<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></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

[Horowitz, “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014]
## Number Representation

<table>
<thead>
<tr>
<th>Number Type</th>
<th>S</th>
<th>E</th>
<th>M</th>
<th>Range</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>1</td>
<td>8</td>
<td>23</td>
<td>$10^{-38} - 10^{38}$</td>
<td>.0000006%</td>
</tr>
<tr>
<td>FP16</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>$6 \times 10^{-5} - 6 \times 10^4$</td>
<td>.05%</td>
</tr>
<tr>
<td>Int32</td>
<td>1</td>
<td></td>
<td>31</td>
<td>$0 - 2 \times 10^9$</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>Int16</td>
<td>1</td>
<td></td>
<td>15</td>
<td>$0 - 6 \times 10^4$</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>Int8</td>
<td>1</td>
<td></td>
<td>7</td>
<td>$0 - 127$</td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>

Image Source: B. Dally
Floating Point $\rightarrow$ Fixed Point

Floating Point

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent (8-bits)</th>
<th>Mantissa (23-bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101101010</td>
<td>0000000000101010000000000100</td>
</tr>
</tbody>
</table>

$-1.42122425 \times 10^{-13}$

$s = 1 \quad e = 70 \quad m = 20482$

Fixed Point

<table>
<thead>
<tr>
<th>Sign</th>
<th>Mantissa (7-bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01100110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integer</th>
<th>Fractional</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bits</td>
<td>3-bits</td>
</tr>
</tbody>
</table>

$12.75$

$s = 0 \quad m = 102$
N-bit Precision

For no loss in precision, $M$ is determined based on largest filter size (in the range of 10 to 16 bits for popular DNNs).
**Dynamic Fixed Point**

**Floating Point**

32-bit float

-1.42122425 x 10^{-13}

```
s = 1  e = 70  m = 20482
```

**Fixed Point**

8-bit dynamic fixed

12.75

```
s = 0  m=102  f = 3
```

8-bit dynamic fixed

0.19921875

```
s = 0  m=102  f = 9
```

Allow f to vary based on data type and layer
Impact on Accuracy

Top-1 accuracy on of CaffeNet on ImageNet

<table>
<thead>
<tr>
<th>Layer outputs</th>
<th>CONV parameters</th>
<th>FC parameters</th>
<th>32-bit floating point baseline</th>
<th>Fixed point accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet (Exp 1)</td>
<td>4-bit</td>
<td>4-bit</td>
<td>4-bit</td>
<td>99.1%</td>
</tr>
<tr>
<td>LeNet (Exp 2)</td>
<td>4-bit</td>
<td>2-bit</td>
<td>2-bit</td>
<td>99.1%</td>
</tr>
<tr>
<td>Full CIFAR-10</td>
<td>8-bit</td>
<td>8-bit</td>
<td>8-bit</td>
<td>81.7%</td>
</tr>
<tr>
<td>SqueezeNet top-1</td>
<td>8-bit</td>
<td>8-bit</td>
<td>8-bit</td>
<td>57.7%</td>
</tr>
<tr>
<td>CaffeNet top-1</td>
<td>8-bit</td>
<td>8-bit</td>
<td>8-bit</td>
<td>56.9%</td>
</tr>
<tr>
<td>GoogLeNet top-1</td>
<td>8-bit</td>
<td>8-bit</td>
<td>8-bit</td>
<td>68.9%</td>
</tr>
</tbody>
</table>

[Gysel et al., Ristretto, ICLR 2016]
Avoiding Dynamic Fixed Point

Batch normalization ‘centers’ dynamic range

‘Centered’ dynamic ranges might reduce need for dynamic fixed point
“New half-precision, 16-bit floating point instructions deliver over 21 TeraFLOPS for unprecedented training performance. With 47 TOPS (tera-operations per second) of performance, new 8-bit integer instructions in Pascal allow AI algorithms to deliver real-time responsiveness for deep learning inference.”

– Nvidia.com (April 2016)
Google’s Tensor Processing Unit (TPU)

“With its TPU Google has seemingly focused on delivering the data really quickly by cutting down on precision. Specifically, it doesn’t rely on floating point precision like a GPU.

....

Instead the chip uses integer math...TPU used 8-bit integer.”

- Next Platform (May 19, 2016)

[Jouppi et al., ISCA 2017]
Precision Varies from Layer to Layer

<table>
<thead>
<tr>
<th>Tolerance</th>
<th>Bits per layer (I+F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet (F=0)</td>
<td>10-8-8-8-8-8-6-4</td>
</tr>
<tr>
<td>1%</td>
<td>10-8-8-8-8-8-5-4</td>
</tr>
<tr>
<td>2%</td>
<td>10-8-8-8-7-7-5-3</td>
</tr>
<tr>
<td>5%</td>
<td>9-8-8-8-7-7-5-3</td>
</tr>
</tbody>
</table>

[Moons et al., WACV 2016]

[Judd et al., ArXiv 2016]
Bitwidth Scaling (Speed)

Bit-Serial Processing: Reduce Bit-width → Skip Cycles
Speed up of 2.24x vs. 16-bit fixed

\[
\sum_{i=0}^{N_i-1} s_i \times n_i = \sum_{i=0}^{N_i-1} s_i \times \sum_{b=0}^{P-1} n_i^b \times 2^b = \sum_{b=0}^{P-1} 2^b \times \sum_{i=0}^{N_i-1} n_i^b \times S_i
\]

[Judd et al., Stripes, CAL 2016]
Bitwidth Scaling (Power)

Reduce Bit-width $\rightarrow$ Shorter Critical Path $\rightarrow$ Reduce Voltage

Power reduction of 2.56x vs. 16-bit fixed On AlexNet Layer 2

$P_{\text{precise}} = \alpha C_f V^2$ $\Rightarrow$ $P_{\text{imprecise}} = \frac{\alpha}{k_1} C_f \left(\frac{V}{k_2}\right)^2$

AlexNet Layer 2 example:

A. 2D-baseline @ 16 bit
B. Precision-Scaling @ 7-7 bit
C. Voltage-Scaling @ 0.9 V
D. Sparse operation guarding

Power reduction of 33x gain @ 1% RMSE

[Moons et al., VLSI 2016]
Binary Nets

• Binary Connect (BC)
  – Weights {-1,1}, Activations 32-bit float
  – MAC $\rightarrow$ addition/subtraction
  – Accuracy loss: 19% on AlexNet
    [Courbariaux, NIPS 2015]

• Binarized Neural Networks (BNN)
  – Weights {-1,1}, Activations {-1,1}
  – MAC $\rightarrow$ XNOR
  – Accuracy loss: 29.8% on AlexNet
    [Courbariaux, arXiv 2016]
Scale the Weights and Activations

**Binary Weight Nets (BWN)**
- Weights \{-\alpha, \alpha\} → except first and last layers are 32-bit float
- Activations: 32-bit float
- \(\alpha\) determined by the \(l_1\)-norm of all weights in a layer
- Accuracy loss: 0.8% on AlexNet

**XNOR-Net**
- Weights \{-\alpha, \alpha\}
- Activations \{-\beta_i, \beta_i\} → except first and last layers are 32-bit float
- \(\beta_i\) determined by the \(l_1\)-norm of all activations across channels for given position \(i\) of the input feature map
- Accuracy loss: 11% on AlexNet

Hardware needs to support both activation precisions

Scale factors (\(\alpha, \beta_i\)) can change per layer or position in filter

[Rastegari et al., BWN & XNOR-Net, ECCV 2016]
XNOR-Net

(1) Binarizing Weight

\[ \frac{1}{n} \|W\|_{\ell_1} = \alpha \]

(2) Binarizing Input

Inefficient

\[ \frac{1}{n} \|X_1\|_{\ell_1} = \beta_1 \]

\[ \frac{1}{n} \|X_2\|_{\ell_1} = \beta_2 \]

Efficient

\[ \sum_{i=1}^{c} \frac{I_{i}}{c} = \sum_{i=1}^{c} \frac{K_{i}}{c} \]

(3) Convolution with XNOR-Bitcount

\[ I \ast W \approx \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \ast \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \]

\[ \alpha \]

[Rastegari et al., BWN & XNOR-Net, ECCV 2016]
Ternary Nets

• Allow for weights to be zero
  – Increase sparsity, but also increase number of bits (2-bits)

• Ternary Weight Nets (TWN) [Li et al., arXiv 2016]
  – Weights \{-w, 0, w\} → except first and last layers are 32-bit float
  – Activations: 32-bit float
  – Accuracy loss: 3.7% on AlexNet

• Trained Ternary Quantization (TTQ) [Zhu et al., ICLR 2017]
  – Weights \{-w_1, 0, w_2\} → except first and last layers are 32-bit float
  – Activations: 32-bit float
  – Accuracy loss: 0.6% on AlexNet
Non-Linear Quantization

• **Precision** refers to the **number of levels**
  - Number of bits = \( \log_2 \) (number of levels)

• **Quantization**: mapping data to a smaller set of **levels**
  - Linear, e.g., fixed-point
  - Non-linear
    • Computed
    • Table lookup

Objective: Reduce size to improve speed and/or reduce energy while preserving accuracy
Computed Non-linear Quantization

Log Domain Quantization

Product = $X \times W$

Product = $X << W$

[Lee et al., LogNet, ICASSP 2017]
Log Domain Computation

(a) Conventional

- Only activation in log domain
- Both weights and activations in log domain

(b) Proposed 1

- Only activation in log domain
- Both weights and activations in log domain

(c) Proposed 2

- Max, bitshifts, adds/subs

[Miyashita et al., arXiv 2016]
Log Domain Quantization

- Weights: 5-bits for CONV, 4-bit for FC; Activations: 4-bits
- Accuracy loss: 3.2% on AlexNet

[Michayluk et al., arXiv 2016],
[Lee et al., LogNet, ICASSP 2017]
Reduce Precision Overview

- Learned mapping of data to quantization levels (e.g., k-means)
  
  [Han et al., ICLR 2016] Implement with look up table

- Additional Properties
  - Fixed or Variable (across data types, layers, channels, etc.)
Non-Linear Quantization Table Lookup

**Trained Quantization:** Find $K$ weights via K-means clustering to reduce number of unique weights *per layer* (weight sharing)

*Example:* AlexNet (no accuracy loss)
- 256 unique weights for CONV layer
- 16 unique weights for FC layer

![Diagram](image)

Consequences: Narrow weight memory and second access from (small) table

[Han et al., Deep Compression, ICLR 2016]
## Summary of Reduce Precision

<table>
<thead>
<tr>
<th>Category</th>
<th>Method</th>
<th>Weights (# of bits)</th>
<th>Activations (# of bits)</th>
<th>Accuracy Loss vs. 32-bit float (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Fixed Point</td>
<td>w/o fine-tuning</td>
<td>8</td>
<td>10</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>w/ fine-tuning</td>
<td>8</td>
<td>8</td>
<td>0.6</td>
</tr>
<tr>
<td>Reduce weight</td>
<td>Ternary weights Networks (TWN)</td>
<td>2*</td>
<td>32</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>Trained Ternary Quantization (TTQ)</td>
<td>2*</td>
<td>32</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>Binary Connect (BC)</td>
<td>1</td>
<td>32</td>
<td>19.2</td>
</tr>
<tr>
<td></td>
<td>Binary Weight Net (BWN)</td>
<td>1*</td>
<td>32</td>
<td>0.8</td>
</tr>
<tr>
<td>Reduce weight and activation</td>
<td>Binarized Neural Net (BNN)</td>
<td>1</td>
<td>1</td>
<td>29.8</td>
</tr>
<tr>
<td></td>
<td>XNOR-Net</td>
<td>1*</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Non-Linear</td>
<td>LogNet</td>
<td>5(conv), 4(fc)</td>
<td>4</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>Weight Sharing</td>
<td>8(conv), 4(fc)</td>
<td>16</td>
<td>0</td>
</tr>
</tbody>
</table>

* first and last layers are 32-bit float

Full list @ [Sze et al., arXiv, 2017]
Reduce Number of Ops and Weights

• Exploit Activation Statistics
• Network Pruning
• Compact Network Architectures
• Knowledge Distillation
Sparsity in Fmaps

Many zeros in output fmaps after ReLU

![Diagram showing sparsity in Fmaps]

<table>
<thead>
<tr>
<th>CONV Layer</th>
<th># of activations</th>
<th># of non-zero activations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6</td>
<td>0.3</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>0.6</td>
</tr>
<tr>
<td>4</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>5</td>
<td>0.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>
I/O Compression in Eyeriss

Run-Length Compression (RLC)

Example:
Input: 0, 0, 12, 0, 0, 0, 0, 53, 0, 0, 22, ...

Output (64b):

RunLevel RunLevel RunLevel RunLevel Term
2 12 4 53 2 22 0

5b 16b 5b 16b 5b 16b 1b

[Chen et al., ISSCC 2016]
Compression Reduces DRAM BW

Simple RLC within 5% - 10% of theoretical entropy limit

[Chen et al., ISSCC 2016]
Data Gating / Zero Skipping in Eyeriss

Skip MAC and mem reads when image data is zero. Reduce PE power by 45%.
Cnvlutin

- Process Convolution Layers
- Built on top of DaDianNao (4.49% area overhead)
- Speed up of 1.37x (1.52x with activation pruning)

[Albericio et al., ISCA 2016]
Pruning Activations

Remove small activation values

**Speed up 11% (ImageNet)**

- **Minerva**
  - Zeros
  - Small Non-zeros

**Reduce power 2x (MNIST)**

- Operations Pruned (%)
- Prediction Error (%)

---

[Albericio et al., ISCA 2016]

[Reagen et al., ISCA 2016]
**Pruning – Make Weights Sparse**

- **Optimal Brain Damage**
  1. Choose a reasonable network architecture
  2. Train network until reasonable solution obtained
  3. Compute the second derivative for each weight
  4. Compute saliencies (i.e. impact on training error) for each weight
  5. Sort weights by saliency and delete low-saliency weights
  6. Iterate to step 2

[Lecun et al., NIPS 1989]
Pruning – Make Weights Sparse

Prune based on magnitude of weights

Example: AlexNet
Weight Reduction: CONV layers 2.7x, FC layers 9.9x
(Most reduction on fully connected layers)
Overall: 9x weight reduction, 3x MAC reduction

[Han et al., NIPS 2015]
Speed up of Weight Pruning on CPU/GPU

On Fully Connected Layers Only
Average Speed up of 3.2x on GPU, 3x on CPU, 5x on mGPU

Intel Core i7 5930K: MKL CBLAS GEMV, MKL SPBLAS CSRMV
NVIDIA GeForce GTX Titan X: cuBLAS GEMV, cuSPARSE CSRMV
NVIDIA Tegra K1: cuBLAS GEMV, cuSPARSE CSRMV

Batch size = 1

[Han et al., NIPS 2015]
Key Metrics for Embedded DNN

• Accuracy → Measured on Dataset
• Speed → Number of MACs
• Storage Footprint → Number of Weights
• Energy → ?
Energy-Aware Pruning

• # of Weights alone is not a good metric for energy
  – Example (AlexNet):
    • # of Weights (FC Layer) > # of Weights (CONV layer)
    • Energy (FC Layer) < Energy (CONV layer)

• Use energy evaluation method to estimate DNN energy
  – Account for data movement

[Yang et al., CVPR 2017]
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 Consumption

Evaluation tool available at http://eyeriss.mit.edu/energy.html
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

DNN Models available at http://eyeriss.mit.edu/energy.html
Energy Estimation Tool

Website: https://energyestimation.mit.edu/

Deep Neural Network Energy Estimation Tool

Overview

This Deep Neural Network Energy Estimation Tool is used for evaluating and designing energy efficient deep neural networks that are critical for embedded deep learning processing. Energy estimation was in the development of the energy-aware pruning method (Yang et al., CVPR 2017), which reduced the energy consumption of AlexNet and GoogleNet by 3.7x and 1.8x, respectively, with less than 1% top-5 accuracy loss. This website provides a simplified version of the energy estimation tool for shorter runtime (around 10 seconds).

Input

To support the variety of toolboxes, this tool takes a single network configuration file. The network configuration file is a text file, where each line denotes the configuration of a CONV/FC layer. The format of each line is:

```
height nChannels nZeroEntries width nMapsOrFilt bits perMap
```

```
2, 27, 27, 96, 44, 3.573e+05, 16, 5, 5, 48, 256, 0, 16, 27, 27, 256, 44, 6.623e+06, 16, 1, 1, 2, 2, 2
```

- **Layer Index:** the index of the layer, from 1 to the number of layers. It should be the same as the line number.
- **Conf IMap, Conf Filt, Conf OMap:** the configuration of the input feature maps, the filters and the output feature maps. The configuration of each of the three data types is in the format of "height width number of channels number of maps or filters number of zero entries bits per map".
- **Stride:** the stride of this layer. It is in the format of "stride, y stride, x".
- **Pad:** the amount of input padding. It is in the format of "pad, top pad, bottom pad, left pad, right pad".

Therefore, there will be 25 entries separated by commas in each line.

Running the Estimation Model

After creating your text file, follow these steps to upload your text file and run the estimation model:

1. Check the "I am not a robot" checkbox and complete the Google reCAPTCHA challenge. Help us prevent spam.
2. Click the "Choose File" button below to choose your text file from your computer.
3. Click the "Run Estimation Model" button below to upload your text file and run the estimation model.

[Yang et al., CVPR 2017]
Compression of Weights & Activations

- Compress weights and activations between DRAM and accelerator
- Variable Length / Huffman Coding

Example:

Value: 16'b0 → Compressed Code: {1'b0}

Value: 16'bx → Compressed Code: {1'b1, 16'bx}

- Tested on AlexNet → 2× overall BW Reduction

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter / Image bits (0%)</th>
<th>Filter / Image BW Reduc.</th>
<th>IO / HuffIO (MB/frame)</th>
<th>Voltage (V)</th>
<th>MMACs/Frame</th>
<th>Power (mW)</th>
<th>Real (TOPS/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General CNN</td>
<td>16 (0%) / 16 (0%)</td>
<td>1.0x</td>
<td>1 / 0.77</td>
<td>1.1</td>
<td>--</td>
<td>288</td>
<td>0.3</td>
</tr>
<tr>
<td>AlexNet 11</td>
<td>7 (21%) / 4 (29%)</td>
<td>1.17x / 1.3x</td>
<td>3.2 / 1.1</td>
<td>0.85</td>
<td>105</td>
<td>85</td>
<td>0.96</td>
</tr>
<tr>
<td>AlexNet 12</td>
<td>7 (19%) / 7 (89%)</td>
<td>1.15x / 5.8x</td>
<td>6.5 / 2.8</td>
<td>0.9</td>
<td>224</td>
<td>55</td>
<td>1.4</td>
</tr>
<tr>
<td>AlexNet 13</td>
<td>8 (11%) / 9 (82%)</td>
<td>1.05x / 4.1x</td>
<td>5.4 / 3.2</td>
<td>0.92</td>
<td>150</td>
<td>77</td>
<td>0.7</td>
</tr>
<tr>
<td>AlexNet 14</td>
<td>9 (04%) / 8 (72%)</td>
<td>1.00x / 2.9x</td>
<td>3.7 / 2.1</td>
<td>0.92</td>
<td>112</td>
<td>95</td>
<td>0.56</td>
</tr>
<tr>
<td>AlexNet 15</td>
<td>9 (04%) / 8 (72%)</td>
<td>1.00x / 2.9x</td>
<td>19.8 / 10</td>
<td>--</td>
<td>--</td>
<td>76</td>
<td>0.94</td>
</tr>
<tr>
<td>Total / avg.</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>LeNet-5 11</td>
<td>3 (35%) / 1 (87%)</td>
<td>1.40x / 5.2x</td>
<td>0.003 / 0.004</td>
<td>0.7</td>
<td>0.3</td>
<td>25</td>
<td>1.07</td>
</tr>
<tr>
<td>LeNet-5 12</td>
<td>4 (26%) / 6 (55%)</td>
<td>1.25x / 1.9x</td>
<td>0.050 / 0.042</td>
<td>0.8</td>
<td>1.6</td>
<td>35</td>
<td>1.75</td>
</tr>
<tr>
<td>Total / avg.</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>33</td>
<td>1.6</td>
</tr>
</tbody>
</table>

[Moons et al., VLSI 2016; Han et al., ICLR 2016]
Sparse Matrix-Vector DSP

- Use CSC rather than CSR for SpMxV

Compressed Sparse Row (CSR)  Compressed Sparse Column (CSC)

Reduce memory bandwidth (when not $M >> N$)

For DNN, $M =$ # of filters, $N =$ # of weights per filter

[Dorrance et al., FPGA 2014]
EIE: A Sparse Linear Algebra Engine

- Process Fully Connected Layers (after Deep Compression)
- Store weights column-wise in Run Length format
- Read relative column when input is non-zero

Supports Fully Connected Layers Only

Input: \( \begin{bmatrix} 0 & a_1 & 0 & a_3 \end{bmatrix} \)

Weights: \( \begin{bmatrix} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & w_2,1 & 0 & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_5,0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & w_{7,1} & 0 & 0 \end{bmatrix} \)

Output: \( \begin{bmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{bmatrix} \)

Dequantize Weight

Keep track of location

Output Stationary Dataflow

[Han et al., ISCA 2016]
Sparse CNN (SCNN)

Supports Convolutional Layers

Input Stationary Dataflow

[Parashar et al., ISCA 2017]
Structured/Coarse-Grained Pruning

- **Scalpel**
  - Prune to match the underlying data-parallel hardware organization for speed up

---

**Example: 2-way SIMD**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Dense weights

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sparse weights

\[
A' = \begin{bmatrix}
(0, 5) & (2, 5) & (1, 7) \\
(2, 3) & (4, 2) & (8, 4)
(8, 3) & (3, 2)
\end{bmatrix}
\]

\[
JA' = \begin{bmatrix}
0 & 2 & 2 & 0 & 4 & 0, 4, 0
\end{bmatrix}
\]

\[
LA' = \begin{bmatrix}
0 & 4 & 6 & 10 & 12 & 14, 16
\end{bmatrix}
\]

Input Vector

[Yu et al., ISCA 2017]
Compact Network Architectures

• Break large convolutional layers into a series of smaller convolutional layers
  – Fewer weights, but same effective receptive field

• Before Training: Network Architecture Design

• After Training: Decompose Trained Filters
Network Architecture Design

Build Network with series of Small Filters

GoogleNet/Inception v3

5x5 filter

5x5 filter

decompose

Apply sequentially

separable filters

5x1 filter

1x5 filter

VGG-16

5x5 filter

5x5 filter

decompose

Apply sequentially

Two 3x3 filters
Network Architecture Design

Reduce size and computation with 1x1 Filter (bottleneck)

Used in Network In Network (NiN) and GoogLeNet

[Lin et al., ArXiV 2013 / ICLR 2014] [Szegedy et al., ArXiV 2014 / CVPR 2015]
Network Architecture Design

Reduce size and computation with 1x1 Filter (bottleneck)

Used in Network In Network (NiN) and GoogLeNet

[Lin et al., ArXiV 2013 / ICLR 2014] [Szegedy et al., ArXiV 2014 / CVPR 2015]
Network Architecture Design

Reduce size and computation with 1x1 Filter (bottleneck)

Used in Network In Network (NiN) and GoogLeNet

Bottleneck in Popular DNN models

ResNet

GoogleNet

compress
expand
Reduce weights by reducing number of input channels by “squeezing” with 1x1 50x fewer weights than AlexNet (no accuracy loss)
Deeper CNNs with fewer weights do not necessarily consume less energy than shallower CNNs with more weights

[Yang et al., CVPR 2017]
Decompose Trained Filters

After training, perform low-rank approximation by applying tensor decomposition to weight kernel; then fine-tune weights for accuracy

\[ \text{Lebedev et al., ICLR 2015} \]

**R** = canonical rank

[Lebedev et al., ICLR 2015]
Decompose Trained Filters

Visualization of Filters

Original   Approx.

- Speed up by 1.6 – 2.7x on CPU/GPU for CONV1, CONV2 layers
- Reduce size by 5 - 13x for FC layer
- < 1% drop in accuracy

[Denton et al., NIPS 2014]
## Decompose Trained Filters on Phone

### Tucker Decomposition

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-5</th>
<th>Weights</th>
<th>FLOPs</th>
<th>S6</th>
<th>Titan X</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>80.03</td>
<td>61M</td>
<td>725M</td>
<td>117ms</td>
<td>245mJ</td>
</tr>
<tr>
<td>AlexNet*</td>
<td>78.33</td>
<td>11M</td>
<td>272M</td>
<td>43ms</td>
<td>72mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-1.70)</td>
<td>(×5.46)</td>
<td>(×2.67)</td>
<td>(×2.72)</td>
<td>(×3.41)</td>
</tr>
<tr>
<td>VGG-S</td>
<td>84.60</td>
<td>103M</td>
<td>2640M</td>
<td>357ms</td>
<td>825mJ</td>
</tr>
<tr>
<td>VGG-S*</td>
<td>84.05</td>
<td>14M</td>
<td>549M</td>
<td>97ms</td>
<td>193mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-0.55)</td>
<td>(×7.40)</td>
<td>(×4.80)</td>
<td>(×3.68)</td>
<td>(×4.26)</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>88.90</td>
<td>6.9M</td>
<td>1566M</td>
<td>273ms</td>
<td>473mJ</td>
</tr>
<tr>
<td>GoogLeNet*</td>
<td>88.66</td>
<td>4.7M</td>
<td>760M</td>
<td>192ms</td>
<td>296mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-0.24)</td>
<td>(×1.28)</td>
<td>(×2.06)</td>
<td>(×1.42)</td>
<td>(×1.60)</td>
</tr>
<tr>
<td>VGG-16</td>
<td>89.90</td>
<td>138M</td>
<td>15484M</td>
<td>1926ms</td>
<td>4757mJ</td>
</tr>
<tr>
<td>VGG-16*</td>
<td>89.40</td>
<td>127M</td>
<td>3139M</td>
<td>576ms</td>
<td>1346mJ</td>
</tr>
<tr>
<td>(imp.)</td>
<td>(-0.50)</td>
<td>(×1.09)</td>
<td>(×4.93)</td>
<td>(×3.34)</td>
<td>(×3.53)</td>
</tr>
</tbody>
</table>

[Kim et al., ICLR 2016]
Knowledge Distillation

[Bucilu et al., KDD 2006],[Hinton et al., arXiv 2015]