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

\[
\begin{array}{ccc}
9 & -1 & -3 \\
1 & -5 & 5 \\
-2 & 6 & -1 \\
\end{array}
\quad \xrightarrow{\text{ReLU}} \quad
\begin{array}{ccc}
9 & 0 & 0 \\
1 & 0 & 5 \\
0 & 6 & 0 \\
\end{array}
\]

- # of activations
- # of non-zero activations

\[\begin{array}{cccc}
\text{CONV Layer} & 1 & 2 & 3 \\
\text{(Normalized)} & 0.6 & 0.8 & 1.0 \\
\end{array}\]

27
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

DCNN Accelerator

Off-Chip DRAM

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

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

[Chen et al., ISSCC 2016]
Skip MAC and mem reads when image data is zero. Reduce PE power by 45%.

[Chen et al., ISSCC 2016]
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)**

**Reduce power 2x (MNIST)**

[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

Memory
Accesses
Optimization

# of MACs
Calculation

# acc. at mem. level 1

# acc. at mem. level 2

... 

# acc. at mem. level n

# of MACs

E_{\text{comp}}

E_{\text{data}}

Energy

L1 L2 L3 ...

CNN 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
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 used 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 1.6x, 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 nMapsOnSlit bitsPerPixel
```

- 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_OutMap: 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 number_of_fills number_of_zero_entries bits_in_bits".
- Stride: the stride of this layer. It is in the format of "stride, x stride, y".
- 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. Click the "I am not a robot" checkbox and complete the Google reCAPTCHA challenge. Helps 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  $\rightarrow$ Compressed Code: {1'b0}

Value: 16'bx  $\rightarrow$ Compressed Code: {1'b1, 16'bx}

• Tested on AlexNet $\rightarrow$ 2× overall BW Reduction

[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
Sparse CNN (SCNN)

**Supports Convolutional Layers**

- Densely Packed Storage of Weights and Activations
- All-to-all Multiplication of Weights and Activations
- Mechanism to Add to Scattered Partial Sums

**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*

<table>
<thead>
<tr>
<th>Dense weights</th>
<th>Sparse weights</th>
<th>Input Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 5 2 5 0 0</td>
<td>0 5 2 5 1 7</td>
<td>0 9 0 1 4 3</td>
</tr>
<tr>
<td>0 0 1 7 0 0</td>
<td>2 3 4 2 4 2</td>
<td></td>
</tr>
<tr>
<td>2 3 0 0 4 2</td>
<td>8 4 8 3 8 3</td>
<td></td>
</tr>
<tr>
<td>8 4 0 0 0 0</td>
<td>3 2 3 2 3 2</td>
<td></td>
</tr>
<tr>
<td>0 0 1 1 8 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 2 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

$JA' = \begin{bmatrix} 0 & 2 & 2 & 0 & 4 & 0 \\ 4 & 0 \end{bmatrix}$

$IA' = \begin{bmatrix} 0 & 4 & 6 & 10 & 12 & 14 & 16 \end{bmatrix}$

[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 → 5x1 filter → 1x5 filter

VGG-16

5x5 filter → Two 3x3 filters

separable filters

Apply sequentially

Mit
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

[Lin et al., ArXiV 2013 / ICLR 2014] [Szegedy et al., ArXiV 2014 / CVPR 2015]
Bottleneck in Popular DNN models

ResNet

GoogleNet

compress
SqueezeNet

Reduce weights by reducing number of input channels by “squeezing” with 1x1 50x fewer weights than AlexNet (no accuracy loss)

[F.N. Iandola et al., ArXiv, 2016]
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.

(a) Full convolution
(b) Two-component decomposition (Jaderberg et al., 2014a)
(c) CP-decomposition

$R = \text{canonical rank}$

[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]