SnaPEA:
Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks

Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K. Gupta, Hadi Esmaeilzadeh

* Equal Contribution

University of California, San Diego
† Georgia Institute of Technology
‡ Qualcomm Technologies, Inc
CNNs perform **trillions** of operations for one input.

<table>
<thead>
<tr>
<th>CNN models</th>
<th>Operations for inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>16,362,000,000,000 Ops</td>
</tr>
<tr>
<td>AlexNet</td>
<td>1,147,000,000,000 Ops</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>283,000,000,000 Ops</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>222,000,000,000 Ops</td>
</tr>
</tbody>
</table>
Convolutions dominate CNN computation

≥ 90% of operations are for convolutional layers
Research challenge:
How to reduce CNN computation with minimal effect on accuracy?

Our solution: SnaPEA
1. Leverage algorithmic structure
2. Exploit runtime information
3. Tune up with static multi-variable optimization
(1) Algorithmic structure of CNNs guides SnaPEA
(1) Algorithmic structure of CNNs guides SnaPEA
Opportunity to reduce the computation

Large number of negative convolution outputs (61% on average)
ReLU makes negative outputs zero: cut convolution short

Blue boxes are the performed operations in two highlighted convolutions

Early termination of convolution
(2) Runtime information enables reducing computation

Rectified Linear Unit (ReLU)

Varying distribution of zero and non-zero outputs
SnaPEA: Principles

SnaPEA: Leveraging algorithmic structure of CNNs and runtime information

- Reduce computation without accuracy loss
- Trade accuracy for further computation reduction
- Add minimal hardware overhead
SnaPEA: An illustrative example

Original convolution

<table>
<thead>
<tr>
<th>w</th>
<th>+</th>
<th>-</th>
<th>+</th>
<th>+</th>
<th>+</th>
<th>-</th>
<th>+</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>+</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>PartialSum</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

ReLU

0
SnaPEA: An illustrative example (Exact mode)

Original convolution

\[
\begin{array}{cccccccccccc}
  x & + & + & + & + & + & + & + & + & + & + & + & + \\
  \text{PartialSum} & + & - & - & + & + & + & + & + & - & - & + & - \\
\end{array}
\]

Convolution in SnaPEA (Exact mode)

\[
\begin{array}{cccccccccccc}
  x & + & + & + & + & + & + & + & + & + & + & + & + \\
  \text{PartialSum} & + & + & + & + & + & + & + & - & - & - & - & - \\
\end{array}
\]
Potential benefits in the exact mode

On average, 54% of the weights are negative
SnaPEA: An illustrative example (Predictive mode)

Convolution in SnaPEA (Predictive mode)
Speculation operations

Large absolute value

n largest weights

Small absolute value

Xw * w ∼ x * W

Group 1 Group 2 Group n

Large

largest weights from each group

Small

Xw * w ∼ x * W
Optimize the level of speculation

Speculation parameters:

- **Th**: Threshold
- **N**: Number of speculation operations

Find (Th, N) for all kernels in a CNN to minimize operations and satisfy the accuracy.
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)
Optimize the level of speculation

All convolution kernels in a CNN:

Layer 1 → Layer 2 → ... → Layer L

Kernel Profiling → Local Optimization → Global Optimization → (Th, N)

Threshold Value vs. # of Operations

Layer 1 → Layer 2 → ... → Layer L

Per kernel sensitivity analysis

Dog
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Threshold Value

# of Operations

Per kernel sensitivity analysis
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Threshold Value

# of Operations

Layer 1

Layer 2

Layer L

Per kernel sensitivity analysis
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Set of configurations per layer
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Set of configurations per layer
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Adjust parameters regarding the cross-layer effect
Optimize the level of speculation

All convolution kernels in a CNN

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Adjust parameters regarding the cross-layer effect
Optimize the level of speculation

All convolution kernels in a CNN

Layer 1  →  Layer 2  →  ...  →  Layer L

Kernel Profiling

Local Optimization

Global Optimization

(Th,N)

Adjust parameters regarding the cross-layer effect
SnaPEA: Hardware implementation

Prediction Activation Unit (PAU)

- **Exact mode**
  - Partial result
  - Terminate

- **Predictive mode**
  - Sign-bit
  - Threshold
  - Terminate

Add low-overhead sign checks and threshold checks to the hardware
SnaPEA: Hardware implementation

Processing Engine (PE)

- Weight and In/Out Buffer
- Index Buffer
- K Compute Lanes
- MAC
- Prediction Activation Unit (PAU)
Experimental setup

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1 Accuracy</td>
<td>57.2%</td>
<td>68.7%</td>
<td>57.5%</td>
<td>70.5%</td>
</tr>
<tr>
<td>Top-5 Accuracy</td>
<td>80.1%</td>
<td>89.0%</td>
<td>80.3%</td>
<td>89.9%</td>
</tr>
</tbody>
</table>

Optimization

- Optimization algorithm built on top of Caffe

Hardware implementation

- Simulation: Cycle accurate
- Power estimation: Design Compiler using TSMC 45 nm
- Baseline design: Eyeriss with the same number MAC units (256)
- SnaPEA area overhead compared to Eyeriss: 4.5%
Experimental results
## Experimental results

### Layers in the predictive mode for accuracy loss $\leq 3\%$

<table>
<thead>
<tr>
<th>Network</th>
<th>% of Conv Layers</th>
<th>Speedup</th>
<th>Energy Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>60.0</td>
<td>2.11×</td>
<td>1.97×</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>84.2</td>
<td>2.14×</td>
<td>2.04×</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>65.4</td>
<td>1.94×</td>
<td>1.84×</td>
</tr>
<tr>
<td>VGGNet</td>
<td>61.5</td>
<td>1.87×</td>
<td>1.73×</td>
</tr>
</tbody>
</table>

On average, **68%** of layers operate in the predictive mode (3% accuracy drop).
Experimental results

Highest speedup (3.6x) in a layer in GoogLeNet
Conclusion

SnaPEA

Exploit algorithmic structure and runtime information
Reduce computations in convolutional layers
Control the accuracy with multi-variable optimization
Add minimal hardware overhead

Future directions

Leverage runtime information (e.g., patterns in inputs and activations)
Expand to other activation functions (e.g., sigmoid)
Tune up the hardware for more parallelism