Quantization Methods for Efficient Neural Networks

Amir Gholami, and
Z. Yao, Z. Dong, S. Kim, Z. Zheng, E. Tan, Q. Huang, Michael Mahoney, Kurt Keutzer, and Pallas Group
University of California Berkeley

Prof. Shao’s class on Hardware for Machine Learning
(EE290, Spring 2021)
• Quantization can help improve the training the speed
AI and Memory Wall

- Quantization can help reduce the memory overhead
AI and Memory Wall

- Quantization can help reduce the memory overhead

---

Amir Gholami, AI and Memory Wall, Blogpost, 2021.
Lower precision Multiply-Acc Has higher Throughput

- Lower precision weights mean less energy per Multiply-Accumulate
- Also enables putting more MAC units per unit of silicon

Big opportunity to enable lower bit precision inference!
Quantization Produces Lower Latency

<table>
<thead>
<tr>
<th>Network</th>
<th>Batch Size 1 FP32</th>
<th>Batch Size 1 INT8</th>
<th>Batch Size 8 FP32</th>
<th>Batch Size 8 INT8</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet v2</td>
<td>1.0</td>
<td>1.9</td>
<td>1.0</td>
<td>4.0</td>
</tr>
<tr>
<td>ResNet50 v1.5</td>
<td>1.0</td>
<td>3.5</td>
<td>1.0</td>
<td>7.3</td>
</tr>
<tr>
<td>VGG-19</td>
<td>1.0</td>
<td>3.1</td>
<td>1.0</td>
<td>7.0</td>
</tr>
<tr>
<td>Inception V4</td>
<td>1.0</td>
<td>4.4</td>
<td>1.0</td>
<td>7.1</td>
</tr>
</tbody>
</table>

- Quantization improves latency, sometimes superlinearly

Results from TensorRT 7.0 (credit: P. Judd)
Quantized arithmetic as compared to FP arithmetic has order of magnitude less:

- **Energy Cost**
- **Area Cost**

```
<table>
<thead>
<tr>
<th>Operation:</th>
<th>Energy (pJ)</th>
<th>Relative Energy Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>16b Add</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>32b Add</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>16b FP Add</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>32b FP Add</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>8b Mult</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>32b Mult</td>
<td>3.1</td>
<td></td>
</tr>
<tr>
<td>16b FP Mult</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>32b FP Mult</td>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>32b SRAM Read (8KB)</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>Operation:</th>
<th>Area (µm²)</th>
<th>Relative Area Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>16b Add</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>32b Add</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>16b FP Add</td>
<td>1360</td>
<td></td>
</tr>
<tr>
<td>32b FP Add</td>
<td>4184</td>
<td></td>
</tr>
<tr>
<td>8b Mult</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td>32b Mult</td>
<td>3495</td>
<td></td>
</tr>
<tr>
<td>16b FP Mult</td>
<td>1640</td>
<td></td>
</tr>
<tr>
<td>32b FP Mult</td>
<td>7700</td>
<td></td>
</tr>
<tr>
<td>32b SRAM Read (8KB)</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
```

"computing’s Energy Problem, M. Horowitz, ISSCC, 2014
(Numbers are rough approximations for 45nm)"
Want to operate in Low Power Regimes to be used in Mobile Devices

- **OZO Digital Pedometer**
  - 80μW, 0.72Wh | 1 year

- **iWatch Series 3**
  - 1.07Wh | 3.8kJ
  - 60mW, 18 hours

- **iWatch**
  - 80μW, 0.72Wh | 1 year

- **iPhone 11 Pro Max**
  - 15Wh = 54kJ
  - Apple: 12h = 4.5 W

- **iPad Pro**
  - 41Wh = 147kJ
  - Apple: 10h use = average 4.1 W

- **13 inch Macbook Air**
  - 54Wh = 194.4kJ
  - Apple: 10h = 7.6 W

- **15 inch Macbook Pro**
  - 76Wh = 273.6kJ
  - Apple: 10h = 7.6 W

- **Eee PC 1000HE**
  - 49Wh = 176kJ
  - Asus: 9.5h = 5.2 W

- **Kindle Oasis**
  - 0.91Wh = 3.276kJ
  - Ebook Friendly: “15days @ 30m/day” = 7.5h @ 0.12 W average

- **Slide**
  - Courtesy of Xiangyu Yu, and Kurt Keutzer
An Integrated Approach to DNN Design

- **Efficient Inference**
  - Large-scale data centers
  - Edge Devices

- **Rapid Training**
  - Need to test candidate NN models quickly and choose the right model

- **NN Architecture Design**
  - Find the right architecture for a target application
An Integrated Approach to DNN Design

- **Efficient Inference**
  - Large-scale data centers
  - Edge Devices

- **Rapid Training**
  - Need to test candidate NN models quickly and choose the right model

- **NN Architecture Design**
  - Find the right architecture for a target application
Outline

- Basic Concepts of Quantization
- Advanced Concepts of Quantization
- Co-design of Neural Network and Hardware
Outline

- Basic Concepts of Quantization
  - Uniform vs Non-Uniform Quantization
  - Symmetric vs Asymmetric Quantization
  - Quantization Granularity: Layer-wise vs Channel-wise
  - Dynamic vs Static Quantization
  - Post Training Quantization vs Quantization Aware Training
• Uniform quantization is a linear mapping from floating point values to quantized integer values.
Uniform Quantization

- $r$: real value
- $r_{\text{max}}, r_{\text{min}}$: max/min range of values
- $B$: Quantization Bits
- $S$ (FP32), $z$ (int): Scale and bias
- $q$: Fixed point quantized values

$$ r = \frac{r_{\text{max}} - r_{\text{min}}}{2^B - 1}(q - z) $$

$$ r = S(q - z) $$

**Floating Point Values**

**8-bit Quantized Values**
Non-Uniform Quantization


Non-uniform quantization is not efficient for hardware deployment.
Symmetric vs Asymmetric Quantization

- How do we choose the range? In general, the range does not have to $[r_{\text{min}}, r_{\text{max}}]$.

$$r = \frac{r_{\text{max}} - r_{\text{min}}}{2^B - 1}(q - z)$$

- Asymmetric Quantization: no constraint on $\beta, \alpha$

$$r = \frac{\beta - \alpha}{2^B - 1}(q - z)$$

- Symmetric Quantization: $\beta = -\alpha$

$$r = \frac{-2\alpha}{2^B - 1}(q - z)$$

- For both cases the range can be different than min/max. A good example, is percentile where we choose to keep a percentage of the values instead of all of them (say 98%).
Quantization Granularity: Layer-wise vs Channel-wise

Layer-wise Quantization:
- Use the same clipping range $[\beta, \alpha]$ for all the convolution kernels in a layer
- Usually sub-optimal since the weights in a layer can have different ranges

Channel-wise Quantization:
- Use the different clipping range $[\beta, \alpha]$ for all the convolution kernels in a layer
- Usually results in better accuracy as the range can be better captured for each output channel
- Has negligible overhead
Dynamic vs Static Quantization

- How do we choose the clipping range $[\beta, \alpha]$? 
  \[ r = \frac{\beta - \alpha}{2^B - 1} (q - z) \]

- For weights, we know the values **statically**, since weights are fixed during inference
- **But what about activations?** We can either use static or dynamic quantization:
  
  - **Static Quantization:** Choose pre-determined static range for activations $(x, y)$, independent of input
    - Very fast, low overhead, but typically not accurate since each input can have a different range
  
  - **Dynamic Quantization:** Determine range for each activation separately during the runtime
    - Typically very slow due to the cost of computing mix/max or percentile
    - But very accurate as it exactly detects the correct range for quantization

\[
\begin{bmatrix}
 y_0 \\
 y_1 \\
 y_2 \\
 y_3 \\
 y_4 \\
\end{bmatrix} = \begin{bmatrix}
 w_{00} & w_{01} & w_{02} \\
 w_{10} & w_{11} & w_{12} \\
 w_{20} & w_{21} & w_{22} \\
 w_{30} & w_{31} & w_{32} \\
 w_{40} & w_{41} & w_{42} \\
\end{bmatrix} \cdot \begin{bmatrix}
 x_0 \\
 x_1 \\
 x_2 \\
\end{bmatrix} \\
 y = W \cdot x
\]
• There is also another classification:

• **Post Training Quantization (PTQ):**
  – Input is a model trained at higher precision after training is finished
  – Quantization is performed by analyzing weights/activations **without fine-tuning**

• **Quantization Aware Training (QAT):**
  – Input is a model trained at higher precision after training is finished
  – Quantization is performed along with possibly **several epochs of fine-tuning**
### Post Training Quantization (PTQ) vs Quantization Aware Training (QAT)

<table>
<thead>
<tr>
<th>PTQ</th>
<th>QAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usually fast</td>
<td>Slow</td>
</tr>
<tr>
<td>No re-training of the model</td>
<td>Model needs to be trained/finetuned</td>
</tr>
<tr>
<td>Plug and play of quantization schemes</td>
<td>Plug and play of quantization schemes (requires re-training)</td>
</tr>
<tr>
<td>Less control over final accuracy of the model</td>
<td>More control over final accuracy since q-params are learned during training.</td>
</tr>
</tbody>
</table>

---

Review

- Basic Concepts of Quantization
  - Uniform vs Non-Uniform Quantization
  - Symmetric vs Asymmetric Quantization
  - Quantization Granularity: Layer-wise vs Channel-wise
  - Dynamic vs Static Quantization
  - Post Training Quantization vs Quantization Aware Training
Outline

- Basic Concepts of Quantization

- Advanced Concepts of Quantization
  - Fake Quantization vs Integer-only Quantization
  - Uniform vs Mixed-Precision Quantization

- Co-design of Neural Network and Hardware
Fake (Simulated) Quantization vs Integer-only/Fixed-Point/Dyadic Quantization

Let’s take a closer look at a layer

\[ \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \\ w_{30} & w_{31} & w_{32} \\ w_{40} & w_{41} & w_{42} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \]

\[ y = W \cdot x \]

Illustration from Sahni Manas
Closer Look at One Layer

For simplicity, let’s consider symmetric quantization => z=0

We first need to accumulate results in INT32 and then rescale back to INT4

\[
\begin{bmatrix}
\gamma_0 \\
\gamma_1 \\
\gamma_2 \\
\gamma_3 \\
\gamma_4
\end{bmatrix} =
\begin{bmatrix}
w_{00} & w_{01} & w_{02} \\
w_{10} & w_{11} & w_{12} \\
w_{20} & w_{21} & w_{22} \\
w_{30} & w_{31} & w_{32} \\
w_{40} & w_{41} & w_{42}
\end{bmatrix} \cdot
\begin{bmatrix}
x_0 \\
x_1 \\
x_2
\end{bmatrix}
\]

\[y = W \cdot x\]

\[y^{i32} = W^{i4} x^{i4}\]

\[y^{i4} = \frac{\alpha_x \alpha_w}{\alpha_y} y^{i32}\]

\[y^{i4} = \text{Round}(S y^{i32})\]

This is Integer-only (aka fixed-point/Dyadic quantization)

But not all quantization algorithms use this method
Fake Quantization!

**Fake Quantization:**

\[
W^{fp_{32}} = (S_w, W^{i4})^{fp_{32}} \quad a^{fp_{32}} = W^{fp_{32}} h^{fp_{32}} \quad a^i = \text{Int}(\frac{a^{fp_{32}}}{S_a})
\]

**Weights**

- INT4
- INT4

**Activations**

\[
h^{fp_{32}} = (S_h, h^{i4})^{fp_{32}}
\]

**FP32 Multiply Accumulate**

**FP32 -> INT4 Requantization**

**Integer-only Quantization**

\[
(S_w, W^{i4}) \quad a^{i32} = W^{i4} h^{i4} \quad a^i = \text{Int}(\frac{S_w S_h}{S_a} a^{i32})
\]

**Weights**

- INT4
- INT4

**Activations**

\[
(S_h, h^{i4})
\]

**INT32 Multiply Accumulate**

**INT32 -> INT4 Dyadic Rescaling**

\[
\frac{S_w S_h}{S_a} = \frac{\alpha}{2^3}
\]
Fake Quantization!

- What is the impact of fake quantization? Are the errors considerable?
- Yes. The errors become quite large for low bit precision.
- Below we report the relative L2 error between integer-only and fake quantization for ResNet50 on ImageNet with INT4 precision.

Relative L2 Error: \[
\frac{\|a - b\|_2}{\|a\|_2}
\]

Layer 4: 1.05%
Layer 24: 58.60%
Layer 50: 97.56%
Last Layer/Logits: 65.50%

Fake Quantization

- In fake quantization weights/activations are converted to FP32 and inference is performed using floating point arithmetic
  - This creates a mismatch which can become significant for low bit integer-only quantization
- Another important difference is that fake quantization uses FP32 for batch normalization layer
  - This is because BatchNorm is sensitive to quantization

We avoid both of these in integer-only quantization (HAWQV3):
- The entire inference is performed with integer-only quantization
- BatchNorm layer is quantized and folded into the convolution weights

https://github.com/zhen-dong/hawq
Integer-only quantization works at 8-bits!

![Table](attachment:image.png)

**Can we go even further and perform lower precision quantization?**


[https://github.com/zhen-dong/hawq](https://github.com/zhen-dong/hawq)
Low Precision Quantization

Can we go even further and perform lower precision quantization?

Uniform low precision does not work as it can significantly degrade accuracy

- Use mixed-precision
• Uniform quantization is a linear mapping from floating point values to quantized integer values.

Flat Loss Landscape $\rightarrow$ Low Bit Precision

- Flat Loss Landscape
- Floating Point values
- 4-bit Quantization
Uniform quantization is a linear mapping from floating point values to quantized integer values.

Sharp Loss Landscape ➔ High Bit Precision Needed

Sharp Loss Landscape

Floating Point values

8-bit Quantization
Using Hessian to Guide Choice of Bit Precision Layer by Layer

This picture misses the hardware. How can we find the right trade-off for a given latency/power constraint for a given hardware platform?
Hardware and Hessian Aware Quantization

Inference Latency

Sensitivity: Flat vs. Sharp Local Minima

Balance the Trade-off

4-bit

8-bit

4-bit

8-bit

4-bit

8-bit

4-bit

8-bit

We find the best bit precision configuration such that:

- Minimally perturbs the model
- Meets application specific:
  - Model size constraint
  - Total bit operations for inference
  - Inference Latency

Objective: \[
\min_{\{b_i\}_{i=1}^L} \sum_{i=1}^L \Omega_i^{(b_i)},
\]

Subject to:
\[
\sum_{i=1}^L M_i^{(b_i)} \leq \text{Model Size Limit},
\]
\[
\sum_{i=1}^L G_i^{(b_i)} \leq \text{BOPS Limit},
\]
\[
\sum_{i=1}^L Q_i^{(b_i)} \leq \text{Latency Limit}.
\]
## Results: ResNet18

(a) *ResNet18*

<table>
<thead>
<tr>
<th>Method</th>
<th>IntOnly</th>
<th>Uniform</th>
<th>Open</th>
<th>Source</th>
<th>Precision</th>
<th>Size (MB)</th>
<th>BOPS (G)</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>–</td>
<td>✓</td>
<td>W32A32</td>
<td>44.6</td>
<td>1858</td>
<td>71.47</td>
<td></td>
</tr>
<tr>
<td>RVQuant [39]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>W8A8</td>
<td>11.1</td>
<td>116</td>
<td>70.01</td>
<td></td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>W8A8</td>
<td>11.1</td>
<td>116</td>
<td><strong>71.56</strong></td>
<td></td>
</tr>
<tr>
<td>PACT [12]</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>W5A5</td>
<td>7.2</td>
<td>50</td>
<td>69.80</td>
<td></td>
</tr>
<tr>
<td>LQ-Nets [54]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>W4A32</td>
<td>5.8</td>
<td>225</td>
<td>70.00</td>
<td></td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>W4/8A4/8</td>
<td>6.7</td>
<td>72</td>
<td>70.22</td>
<td></td>
</tr>
<tr>
<td>HAWQV3+DIST</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>W4/8A4/8</td>
<td>6.7</td>
<td>72</td>
<td><strong>70.38</strong></td>
<td></td>
</tr>
<tr>
<td>CalibTIB [26]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>W4A4</td>
<td>5.8</td>
<td>34</td>
<td>67.50</td>
<td></td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>W4A4</td>
<td>5.8</td>
<td>34</td>
<td><strong>68.45</strong></td>
<td></td>
</tr>
</tbody>
</table>
## Results: ResNet50

### (b) ResNet50

<table>
<thead>
<tr>
<th>Method</th>
<th>IntOnly</th>
<th>Uniform</th>
<th>Open</th>
<th>Source</th>
<th>Bit</th>
<th>Precision</th>
<th>Size (MB)</th>
<th>BOPS (G)</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>–</td>
<td>✗</td>
<td></td>
<td></td>
<td>W32A32</td>
<td>97.8</td>
<td>3951</td>
<td>77.72</td>
</tr>
<tr>
<td>Integer Only [29]</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td></td>
<td></td>
<td>W8A8</td>
<td>24.5</td>
<td>247</td>
<td>74.90</td>
</tr>
<tr>
<td>RVQuant [39]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td>W8A8</td>
<td>24.5</td>
<td>247</td>
<td>75.67</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>W8A8</td>
<td>24.5</td>
<td>247</td>
<td><strong>77.58</strong></td>
</tr>
<tr>
<td>PACT [12]</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td></td>
<td></td>
<td>W5A5</td>
<td>16.0</td>
<td>101</td>
<td>76.70</td>
</tr>
<tr>
<td>LQ-Nets [54]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td></td>
<td></td>
<td>W4A32</td>
<td>13.1</td>
<td>486</td>
<td>76.40</td>
</tr>
<tr>
<td>RVQuant [39]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td>W5A5</td>
<td>16.0</td>
<td>101</td>
<td>75.60</td>
</tr>
<tr>
<td>HAQ [48]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td></td>
<td></td>
<td>WMPA32</td>
<td>9.62</td>
<td>520</td>
<td>75.48</td>
</tr>
<tr>
<td>OneBitwidth [11]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>W1*A8</td>
<td>12.3</td>
<td>494</td>
<td>76.70</td>
</tr>
<tr>
<td>HAWQV3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>W4/8A4/8</td>
<td>18.7</td>
<td>154</td>
<td><strong>75.39</strong></td>
</tr>
<tr>
<td>HAWQV3+DIST</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>W4/8A4/8</td>
<td>18.7</td>
<td>154</td>
<td><strong>76.73</strong></td>
</tr>
</tbody>
</table>

https://github.com/zhen-dong/hawq
How Does ILP Find the Trade-Off?

(a) ResNet18

<table>
<thead>
<tr>
<th>Latency</th>
<th>Size (MB)</th>
<th>BOPS (G)</th>
<th>Speed</th>
<th>Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT8</td>
<td>–</td>
<td>11.2</td>
<td>114</td>
<td>1x</td>
</tr>
<tr>
<td>High</td>
<td>8.7</td>
<td>92</td>
<td>1.12x</td>
<td>70.40/71.05</td>
</tr>
<tr>
<td>Medium</td>
<td>7.2</td>
<td>76</td>
<td>1.19x</td>
<td>70.34/70.55</td>
</tr>
<tr>
<td>Low</td>
<td>6.1</td>
<td>54</td>
<td>1.35x</td>
<td>68.56/69.72</td>
</tr>
<tr>
<td>INT4</td>
<td>–</td>
<td>5.6</td>
<td>28</td>
<td>1.48x</td>
</tr>
</tbody>
</table>
Does this only work for CV tasks?

Integer-only BERT

- We can also use integer-only quantization for BERT
  - Similar to vision, current BERT quantization uses fake quantization
  - Parameters are stored as integer, however operations are processed in floating point

Simulated Quantization (Q-BERT)  Simulated Quantization (Q8BERT)  Integer-only Quantization (I-BERT)

S. Kim, A. Gholami, Z. Yao, M. Mahoney, K. Keutzer, I-BERT: Integer-only BERT Quantization, work in progress.
## Integer-only BERT

- Dyadic quantization works for transformers!

### (a) RoBERTa-Base

<table>
<thead>
<tr>
<th></th>
<th>Int-only</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>×</td>
<td>87.8</td>
<td>87.4</td>
<td>90.4</td>
<td>92.8</td>
<td>94.6</td>
<td>61.2</td>
<td>91.1</td>
<td>90.9</td>
<td>78.0</td>
<td>86.0</td>
</tr>
<tr>
<td>I-BERT</td>
<td>✓</td>
<td>87.5</td>
<td>87.4</td>
<td>90.2</td>
<td>92.8</td>
<td>95.2</td>
<td>62.5</td>
<td>91.1</td>
<td>91.1</td>
<td>79.4</td>
<td>86.3</td>
</tr>
<tr>
<td>Diff</td>
<td></td>
<td>-0.3</td>
<td>0.0</td>
<td>-0.2</td>
<td>0.0</td>
<td>+0.6</td>
<td>+1.3</td>
<td>-0.3</td>
<td>+0.2</td>
<td>+1.4</td>
<td>+0.3</td>
</tr>
</tbody>
</table>

### (b) RoBERTa-Large

<table>
<thead>
<tr>
<th></th>
<th>Int-only</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>×</td>
<td>90.0</td>
<td>89.9</td>
<td>92.8</td>
<td>94.1</td>
<td>96.3</td>
<td>68.0</td>
<td>92.2</td>
<td>91.8</td>
<td>86.3</td>
<td>89.0</td>
</tr>
<tr>
<td>I-BERT</td>
<td>✓</td>
<td>90.4</td>
<td>90.3</td>
<td>93.0</td>
<td>94.5</td>
<td>96.4</td>
<td>69.0</td>
<td>92.2</td>
<td>93.0</td>
<td>87.0</td>
<td>89.5</td>
</tr>
<tr>
<td>Diff</td>
<td></td>
<td>+0.4</td>
<td>+0.4</td>
<td>+0.2</td>
<td>+0.4</td>
<td>+0.1</td>
<td>+1.0</td>
<td>0.0</td>
<td>+1.2</td>
<td>+0.7</td>
<td>+0.5</td>
</tr>
</tbody>
</table>
Review

- Basic Concepts of Quantization

- Advanced Concepts of Quantization
  - Fake Quantization vs Integer-Only Quantization
  - Uniform vs Mixed-Precision Quantization

- Co-design of Neural Network and Hardware
Outline

- Basic Concepts of Quantization
- Advanced Concepts of Quantization
- Co-design of Neural Network and Hardware
An Integrated Approach to DNN Design

NN Architecture Design
- Find the right architecture for a target application

Rapid Training
- Need to test candidate NN models quickly and choose the right model

Efficient Inference
- Large-scale data centers
- Edge Devices
Arithmetic Intensity (AI)

- #Params and MAC could be misleading
  - Ignores memory movement cost
- The right metric is **Arithmetic Intensity**

\[ AI = \frac{\text{FLOP}}{\text{memory operations}} \]


Good video by Sam Williams: https://www.youtube.com/watch?v=hX8KjB3fJ3M
Arithmetic Intensity (AI)

D. Patterson, CACM October 2004

Not All Layers Have the Same Arithmetic Intensity

- **Not All Layers Have the Same Arithmetic Intensity**

  - **Graphs** showing speed-up for different channel sizes:
    - Horizontal graph: Speed-up for 4x4, 8x8, 16x16, 32x32, 64x64, 128x128.
    - Vertical graph: Speed-up for 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024.

- **Diagram** illustrating the effect of channel size on arithmetic intensity.

- **Equations** and **notations**:
  - $H \times W$ and $C$ for dimensions.
  - $K \times K$ for kernel size.
  - $C_{in}$ and $C_{out}$ for input and output channels.

- **Key Points**:
  - Different layers have varying arithmetic intensity.
  - Speed-up is a function of channel size.
  - Optimization opportunities exist for different layer types.

- **Conclusion**:
  - Layer-specific optimizations are essential for efficient computation.

---

© Amir Gholami, UCB Berkeley EE290, 2021
• In collaboration with Samsung, we were able to get HW metrics for each layer so that we can also change the NN architecture so that it would run optimally on the HW

=> Hardware-aware NN Design

MobileNet

Per-Layer Inference Performance

DepthWise Separable Convolution

- Dense 3x3 convolution

(Image credit Yin Guobing)
• DepthWise (DW) 3x3 convolution
  – AI reduced by a factor of 3

(Image credit Yin Guobing)
SqueezeNext Micro-Architecture

Idea:

• Instead of using DW Conv, let’s use Spatial Separable Convs

• **Manually** adapt the NN architecture based on HW metrics

---

• Some layers have poor hardware utilization
  – Redesign the macro-architecture based on hardware utilization

### SqueezeNext Accuracy with Squeezelator Optimization

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1</th>
<th>Top-5</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>57.10</td>
<td>80.30</td>
<td>60.9M</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>57.50</td>
<td>80.30</td>
<td>1.2M</td>
</tr>
<tr>
<td>1.0-SqNxt-23</td>
<td>59.05</td>
<td>82.60</td>
<td>0.72M</td>
</tr>
<tr>
<td>1.0-G-SqNxt-23</td>
<td>57.16</td>
<td>80.82</td>
<td>0.54M</td>
</tr>
<tr>
<td>MobileNet</td>
<td>67.50(70.9)</td>
<td>86.59(89.9)</td>
<td>4.2M</td>
</tr>
<tr>
<td>2.0-Sqxt-23v5</td>
<td>67.44(69.8)</td>
<td>88.20(89.5)</td>
<td>3.2M</td>
</tr>
</tbody>
</table>

- Matches AlexNet with **112x** smaller parameters
- Deeper version achieves VGG accuracy with **36x** smaller model
- Exceeds MobileNet’s top-5 by **+1.6%** with **1.3x** fewer parameters
But Manual Design is not Scalable

- Manual design:
  - Each iteration to evaluate a point in the design space is very expensive
  - Exploration limited by human imagination


Slide: Courtesy of Bichen Wu, and Kurt Keutzer
Can we automate this?

• Manual design:
  • Each iteration to evaluate a point in the design space is very expensive
  • Exploration limited by human imagination


Slide: Courtesy of Bichen Wu, and Kurt Keutzer
Differentiable Neural Architecture Search

Search space

Stochastic super net

Target device

Deploy

Benchmark

Operator Latency

LUT

Proxy dataset

$\mathcal{L}(w, \theta)$

Loss function

Operators

Probability

Sample

Neural Architectures


Slide: Courtesy of Bichen Wu, and Kurt Keutzer
Design Study 1: A11 vs Snapdragon 835

**Apple A11**
- Big: 2 ARMv8 @ 2.5 GHz
- Little: 4 ARMv8 @ 1.4 GHz
- Vectorization: 4-wide 32-bit MAC
- LPDDR4x memory (30 GB/s)
- GPU + Neural Processing Engine

**Snapdragon 835**
- Big: 4 ARMv8 @ 2.4 GHz
- Little: 4 ARMv8 @ 1.9 GHz
- Vectorization: 4-wide 32-bit MAC
- LPDDR4x memory (30 GB/s)
- Adreno 540 GPU


Slide: Courtesy of Bichen Wu, and Kurt Keutzer
Result: FBNet for different target devices

- Apple A11
  - Big: 2 ARMv8 @ 2.5 GHz
  - Little: 4 ARMv8 @ 1.4 GHz
  - Vectorization: 4-wide 32-bit MAC
  - LPDDR4x memory (30 GB/s)
  - GPU + Neural Processing Engine

- Snapdragon 835
  - Big: 4 ARMv8 @ 2.4 GHz
  - Little: 4 ARMv8 @ 1.9 GHz
  - Vectorization: 4-wide 32-bit MAC
  - LPDDR4x memory (30 GB/s)
  - Adreno 540 GPU

- Under similar accuracy constraint (73.27% vs 73.20%), FBNet optimized for iPhone-X achieves 1.4x speedup over the Samsung optimized model


Slide: Courtesy of Bichen Wu, and Kurt Keutzer
Can We Co-Design the Hardware as Well?

- An interesting next step would be to couple Neural Architecture Search with hardware design.
- It may be possible to then perform a large scale search to find an architecture that has good performance for a range of tasks.
- This requires a fast, and accurate hardware simulator.
  - TimeLoop is a recent work enabling this.

Summary: Three Elements of Efficiency at the Edge

New DNN Models

Optimizations:
- Pruning
- Quantization
- Distillation

New Processors And DNN Accelerators
Summary

• Not all layers have the same hardware utilization
  – Pointwise convolution typically achieves high utilization
  – DepthWise convolution typically achieves very low utilization
• FLOPS or #Params is not the correct metric to measure efficiency
  – The right metric is **Arithmetic Intensity** which is **hardware specific**
• The next milestone is to co-design of NN and hardware
  – SqueezeNext was an early work followed by automated DNAS but much more work is left to do
Thanks for Listening

For any feedback/questions please contact amirgh@berkeley.edu