EFFICIENT NN W/O MULTIPLIERS &
ML-ENABLED RISC-V FOR NN

Sebastian Vogel
NXP - CTO Automotive System Innovations
NOVEMBER 2021
EFFICIENT NNS WITHOUT MULTIPLIERS AND ML-ENABLED RISC-V FOR NN

OVERVIEW

• Quantization of Neural Networks w/o Multipliers
  - Self-supervised quantization of pre-trained DNNs
  - Stochastic binary quantization of DNNs
  - Logarithmic quantization at arbitrary base
  - Bit-shift based quantization

• ML-enabled RISC-V
  - Accessibility
  - Spotlight for quantized DNN processing on RISC-V
Quantization of DNNs w/o Multipliers

Bitwise neural networks

Self-supervised quantization

Logarithmic number representation
STOCHASTIC ROUNING DURING TRAINING IMPROVES THE ACCURACY OF BITWISE
NEURAL NETWORKS

- Hubara et al.\cite{1} proposed binary neural networks where weights and activations are
  restricted to $\pm 1$ thereby allowing hardware implementations w/o the need for multipliers
  - The following function for binarizing activation values could also be used for weights
    \[
    \sigma(x) = \begin{cases} 
    +1 & \text{if } x \geq 0 \\
    -1 & \text{otherwise}
    \end{cases}
    \]
  - Instead, the authors suggest using a stochastic rounding procedure for weights
    \[
    sround(w) = \begin{cases} 
    \lceil w \rceil, \text{ with probability } p = \frac{|w|-w}{|w|-w} \\
    \lfloor w \rfloor, \text{ with probability } 1-p = \frac{|w|-w}{|w|-w}
    \end{cases}
    \]
    - Here, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the ceiling and flooring function, respectively,
      mapping inputs to the next valid value. In case of binary networks, these are $\pm 1$.
    - This rounding method preserves the weight value $w$ in the expected value: 
      $E(sround(w)) = w$

- Deterministic and stochastic rounding of weights during training resulted in float32
  networks that achieved 9.9% and 8.27% error rate on CIFAR10, respectively\cite{2}.

\cite{1} Hubara et al., Binarized Neural Networks: Training Neural Networks with Weights and
Activations Constrained to $+1$ or $-1$, NIPS 2016
\cite{2} Courbariaux et al., BinaryConnect: Training Deep Neural Networks with binary weights during
propagations, NIPS 2015
STOCHASTIC Rounding of weights during test-time inference achieves higher accuracy than the model with weights in float32 format

- Vogel et al. suggested using the stochastic rounding of weights also at test-time inference

\[
sround(w) = \begin{cases} 
[w], & \text{with probability } p = \frac{\lfloor w \rfloor - w}{\lfloor w \rfloor - \lfloor w \rfloor} \\
\lfloor w \rfloor, & \text{with probability } 1 - p = \frac{\lfloor w \rfloor - w}{\lfloor w \rfloor - \lfloor w \rfloor}
\end{cases}
\]

- Thereby, creating an ensemble of binarized networks (weights & activations) from a single trained baseline

- With this approach, an ensemble of bitwise networks achieves higher accuracy than the float32 parent

---


STOCHASTIC ROUNCING OF WEIGHTS DURING TEST-TIME INFERENCE ACHIEVES HIGHER ACCURACY THAN THE MODEL WITH WEIGHTS IN FLOAT32 FORMAT

• One hypothesis for this phenomenon (higher accuracy than the float32 model) is that stochastic rounding behaves similar as test-time data augmentation\(^5\)

• The following experiment shows indeed higher accuracy of the float32 model with data augmentation by additive random noise \((n \sim N(0, 0.016^2))\)

![GTSRB Classification](image)

[5] Vogel et al., Efficient Hardware Acceleration for Approximate Inference of Bitwise Deep Neural Networks, DASIP 2017
A simple multiplexer may determine $sround(w)$ efficiently within a single clock cycle.

- When choosing $sel \sim U(1,N)$, $P(out = 1)$ is determined by the number of ‘ones’ in $in$

$$P(out = 1) = \sum_{i=1}^{N} in_i \cdot P(sel = i) = \sum_{i=1}^{N} in_i \frac{1}{N}$$

- Instead, for $P(sel = i) = \frac{2^{N-i}}{2^{N} - 1}$ and $in = \sum_{i=1}^{N} 2^{-1}in_i \in [0,1)$,

$$P(out = 1) = \sum_{i=1}^{N} in_i \cdot P(sel = i) \propto in$$

- $P(sel = i) = \frac{2^{N-i}}{2^{N} - 1}$ can be achieved by:

$$P(sel_j = 1) = \frac{1}{2^{2j-1} + 1}$$

and implemented by:


FEW-BIT QUANTIZATION WITH ARBITRARY LOG-BASE IS A PROMISING APPROACH FOR PRESERVING PRE-TRAINED NETWORK ACCURACY

• As of 2018, few-bit-quantization lacked behind sota floating point training and resulted in complex training routines and hard to master training “ingredients”

• Quantization of pre-trained DNNs favorable

• Let’s first have a look at integer quantization and a suitable quantization error metric

Quantization of DNNs (w/o Multipliers)

Bitwise neural networks

Self-supervised quantization

Logarithmic number representation
SELF-SUPERVISED QUANTIZATION OF PRE-TRAINED NEURAL NETWORKS DOES NOT REQUIRE LABELLED TRAINING DATA

- Quantizing pre-trained neural networks, i.e., determining the quantization step size $\alpha$
  - Without the need for labeled training data through self-supervised quantization$^7$
  - Unlabeled calibration enough

\[
\text{quant}(\cdot): y \mapsto y_q = \alpha \cdot \text{clip}\left(\text{round}\left(\frac{y}{\alpha}\right), -2^{N-1}, 2^{N-1} - 1\right)
\]

\[
y^{(l)} = \Phi \left( b^{(l)} + \sum w^{(l)} x^{(l)} \right)
\]

\[
y_q^{(l)} = \text{quant}(y^{(l)}, \alpha) = y^{(l)} + y^{(l)}_{\Delta} \underbrace{\text{QE}}_{\text{propQE}}
\]

Option 1: Minimize the squared QE\[
\alpha = \text{argmin}\left(y^{(l)}_{\Delta}^2\right)
\]

Option 2: Minimize squared propagated quantization error\[
\alpha = \text{argmin}\left(y^{(l)}_{p\Delta}^2\right)
\]

$^7$ Vogel et al., Self-Supervised Quantization of Pre-Trained Neural Networks for Multiplierless Acceleration, DATE 2019
SELF-SUPERVISED QUANTIZATION OF PRE-TRAINED NEURAL NETWORKS

• 8bit quantization (per-tensor) of activations only

<table>
<thead>
<tr>
<th>Quantization</th>
<th>VGG16 top-1⁺ top-5++</th>
<th>ResNet50 top-1 top-5</th>
<th>InceptionNet top-1 top-5</th>
<th>Dilated Model mIoU§ pix.acc.#</th>
<th>FCN8s mIoU pix.acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration samples</td>
<td>100</td>
<td></td>
<td></td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>Float32 baseline</td>
<td>69.58 89.04</td>
<td>72.99 90.93</td>
<td>75.61 92.48</td>
<td>55.63 92.85</td>
<td>66.48 94.65</td>
</tr>
</tbody>
</table>

Semantics Segmentation

Classification

• 8bit quantization (per-tensor) of activations only

- Float 32bit
- Linear 8bit (params & act.)


* Top-1 accuracy: % of correctly classified labels
** Top-5 accuracy: % of correct label within first 5 predicted labels
§ mIoU: mean intersection over union
# pix.acc.: mean overall pixel accuracy
SELF-SUPERVISED QUANTIZATION OF PRE-TRAINED NEURAL NETWORKS

- 8bit quantization (per-tensor) of activations only

<table>
<thead>
<tr>
<th>Quantization</th>
<th>VGG16</th>
<th>ResNet50</th>
<th>InceptionNet</th>
<th>Dilated Model</th>
<th>FCN8s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration samples</td>
<td>100</td>
<td>36</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Float32 baseline</td>
<td>69.58</td>
<td>89.04</td>
<td>72.99</td>
<td>75.61</td>
<td>66.48</td>
</tr>
<tr>
<td>$y_q$ max abs (naïve)</td>
<td>66.36</td>
<td>88.82</td>
<td>64.75</td>
<td>0.00</td>
<td>64.68</td>
</tr>
<tr>
<td>$y_q$ min MSE (Opt. 1)</td>
<td>68.51</td>
<td>88.79</td>
<td>70.08</td>
<td>69.66</td>
<td>65.04</td>
</tr>
<tr>
<td>$y_q$ min propQE (Opt. 2)</td>
<td>69.09</td>
<td>88.97</td>
<td>71.31</td>
<td>73.89</td>
<td>66.49</td>
</tr>
<tr>
<td>propQE vs baseline</td>
<td>-0.49</td>
<td>-0.07</td>
<td>-1.68</td>
<td>-1.72</td>
<td>+0.01</td>
</tr>
</tbody>
</table>

Float 32bit                                    Linear 8bit (params & act.)


[4] [Image]

$\text{top-1}^+$: Top-1 accuracy
$\text{top-5}^{++}$: Top-5 accuracy
$m\text{IoU}$: mean intersection over union
$\text{pix.acc.}$: mean overall pixel accuracy

* Top-1 accuracy: % of correctly classified labels
** Top-5 accuracy: % of correct label within first 5 predicted labels
§ mIoU: mean intersection over union
# pix.acc.: mean overall pixel accuracy
Quantization of DNNs w/o Multipliers

Bitwise neural networks

Self-supervised quantization

Logarithmic number representation
FEW-BIT QUANTIZATION WITH ARBITRARY LOG-BASE IS A PROMISING APPROACH FOR PRESERVING PRE-TRAINED NETWORK ACCURACY

• As of 2018, few-bit-quantization lacked behind sota floating point training and resulted in complex training routines and hard to master training “ingredients”

• Quantization of pre-trained DNNs favorable

• CNN accelerators incorporate a considerable amount of multiply-accumulate (MAC) engines

• Reducing the bit-widths optimizes for power and memory requirements

• Adders and bit-shifts lead to considerably reduced area requirements compared to MACs

\[ \log_2(x) \cdot \log_2(w) \]

Logarithmic quantization incorporates an intrinsic pruning effect when choosing base $a < 2$ \[^8\]

$$a \in \left\{2^{2^{-\hat{a}}} \mid \hat{a} \in \mathbb{N}_0\right\}$$

$$x \cdot w =$$

LUT w/ $2^{\hat{a}}$ entries

- Logarithmic quantization incorporates an intrinsic pruning effect when choosing base $a < 2$ \[^8\]

---

\[^8\] Vogel et al., Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base, ICCAD 2018
\[^4\] Vogel, Design and implementation of number representations for efficient multiplierless acceleration of convolutional neural networks, PhD Thesis 2020
THE SAME OPTIMAL LOG-BASE IS FOUND FOR ALL LAYERS, MAKING A HW-IMPLEMENTATION LESS COMPLEX

- Optimal log-bases are determined by minimizing the propagated quantization error (propQE)
- Different optimal log-bases are found for weights and activations
- For ResNet50, the same optimal log-base is found in every layer
  → No HW-flexibility required for changing the log-base

---

In ResNet50, the same optimal log-base is found in every layer. In InceptionResNet, there are exceptions to this behavior, yet choosing a single optimal log-base for all layers achieves still considerably good results.
**LOG-BASED QUANTIZATION ACHIEVES COMPETITIVE RESULTS COMPARED TO LINEAR QUANT. ON SEVERAL DNN ARCHITECTURES**

- Logarithmic quantization of weights* and activations at 5 bit

### Classification

<table>
<thead>
<tr>
<th>Quantization</th>
<th>Bit-Width</th>
<th>VGG16</th>
<th>ResNet50</th>
<th>InceptionNet</th>
<th>Dilated Model</th>
<th>FCN8s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration samples</td>
<td>–</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lin-quant baseline</td>
<td>8</td>
<td>69.12</td>
<td>89.06</td>
<td>71.67</td>
<td>90.73</td>
<td>73.71</td>
</tr>
</tbody>
</table>

### Semantic Segmentation

- mIoU: mean intersection over union
- pix.acc.: mean overall pixel accuracy

### Logarithmic 5bit vs. Linear 8bit

- Logarithmic 5bit
- Linear 8bit

---

* per-tensor quantization and biases @8bit (linear) per-tensor

Log-based MAC-elements are complex but have reduced interface bit-widths.

- Log-based number representations allow reducing the external bit-widths and therefore, optimize external bus and memory requirements.
- Nevertheless, an implementation of a log-based MAC*-element consists of more stages than its linear implementation.

* MAC – multiply-accumulate

ARE THERE WAYS TO ADDRESS THE DISCUSSED DOWNSIDES OF THIS LOG-BASED NUMBER REPRESENTATION?

• In the following, an alternate approach is presented addressing the drawbacks of log-based quantization with arbitrary log-base
  - Complex MAC-element implementation
  - Reduced accuracy on complex DNN architectures

<table>
<thead>
<tr>
<th>Quantization</th>
<th>Bit Width</th>
<th>VGG16 top-1</th>
<th>VGG16 top-5</th>
<th>ResNet50 top-1</th>
<th>ResNet50 top-5</th>
<th>InceptionNet top-1</th>
<th>InceptionNet top-5</th>
<th>Dilated Model mIoU pix.acc.</th>
<th>FCN8s mIoU pix.acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration samples</td>
<td>–</td>
<td>100</td>
<td></td>
<td>36</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lin-quant baseline</td>
<td>8</td>
<td>69.12</td>
<td>69.06</td>
<td>71.67</td>
<td>90.73</td>
<td>73.71</td>
<td>91.57</td>
<td>55.62</td>
<td>92.78</td>
</tr>
<tr>
<td>w: log2(10), y: log2(x)</td>
<td>5</td>
<td>68.46</td>
<td>88.36</td>
<td>66.89</td>
<td>87.08</td>
<td>64.65</td>
<td>85.55</td>
<td>54.83</td>
<td>92.65</td>
</tr>
<tr>
<td>log vs linear</td>
<td>–</td>
<td>-0.66</td>
<td>-0.70</td>
<td>-4.78</td>
<td>-3.65</td>
<td>-9.06</td>
<td>-6.02</td>
<td>-0.79</td>
<td>-0.13</td>
</tr>
</tbody>
</table>

**LOG-BASED MIXED-PRECISION QUANTIZATION ADDRESSES SIMPLER IMPLEMENTATION AND HIGHER ACCURACY ON COMPLEX DNN ARCHITECTURES**

- CNN accelerators incorporate a considerable amount of multiply-accumulate engines
- Fixed-point multipliers are considerably larger (wrt. silicon area) than shift-operations

  - Shift-based operation
    → logarithmically quantized weights (4bit)

  - Note:
    This approach uses linearly quantized activations and therefore, integrates standard input signals more easily
LOG-BASED MIXED-PRECISION QUANTIZATION ADDRESSES SIMPLER IMPLEMENTATION AND HIGHER ACCURACY ON COMPLEX DNN ARCHITECTURES

\[ w_1, w_2 \in \mathbb{Z} \{ 2^z \mid z \in \mathbb{N}_0 \} \]

\[ x \cdot (w_1 + w_2) \]

\[ x \ll \log_2(w_1) + x \ll \log_2(w_2) \]

• Quantization of weights (with bimodal distribution)
  • linear
  • “one-hot”
  • “two-hot”

Log-based quantization (per-tensor) of weights, biases*, and activations*

Log-based quantization achieves competitive results compared to linear quant. even on complex DNN architectures

<table>
<thead>
<tr>
<th>Quantization</th>
<th>VGG16 top-1* top-5**</th>
<th>ResNet50 top-1 top-5</th>
<th>InceptionNet top-1 top-5</th>
<th>Dilated Model mIoU§ pix.acc.#</th>
<th>FCN8s mIoU pix.acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calibration samples</td>
<td>100</td>
<td></td>
<td></td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>lin-quant baseline</td>
<td>69.12  89.06</td>
<td>71.67  90.73</td>
<td>73.71  91.57</td>
<td>55.62  92.78</td>
<td>66.47  94.44</td>
</tr>
<tr>
<td>𝑤𝑤𝑞𝑞 one-hot, 4 bit</td>
<td>63.85  86.76</td>
<td>46.36  72.11</td>
<td>37.77  64.55</td>
<td>49.52</td>
<td></td>
</tr>
<tr>
<td>𝑤𝑤𝑞𝑞 two-hot, 8 bit</td>
<td>68.91  89.54</td>
<td>70.84  90.35</td>
<td>72.47  91.11</td>
<td>55.34  92.74</td>
<td>66.24  94.41</td>
</tr>
<tr>
<td>Two-hot vs linear</td>
<td>-0.21 +0.48 -0.83 -0.38 -1.24 -0.46 -0.28 -0.04 -0.23 -0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Top-1 accuracy: % of correctly classified labels
- Top-5 accuracy: % of correct label within first 5 predicted labels
- mIoU: mean intersection over union
- pix.acc.: mean overall pixel accuracy

• Layers close to the network input are sensitive to one-hot quantization
• Layerwise selection allows to trade accuracy with throughput and resulting network size
• The configuration can be selected at run-time

Mixed-precision log-based quantization allows to trade accuracy with throughput and network size [9]

[9] Vogel et al., Bit-Shift-Based Accelerator for CNNs with Selectable Accuracy and Throughput, DSD 2019
BIT-SHIFT-BASED MAC-ELEMENTS WITH LINEAR QUANTIZATION FOR ACTIVATIONS OFFER FLEXIBLE MIXED-PRECISION COMPUTATION

• Implementations of bit-shift-based MAC*-elements with “one-hot”/”two-hot” weights are less complex than log-based MAC-elements with arbitrary log-base
• Mixed-precision capability built in without the need for upper/lower nibble** handling

* MAC – multiply-accumulate
** nibble – 4 bit

QUALITATIVE EVALUATION ON SEMANTIC SEGMENTATION

- Qualitative output of the dilated model for semantic segmentation on cityscapes
- Linear 8bit quantization (left), two-hot 8bit quantization (right), mutual diff. (bottom)

A METHOD FOR QUANTIZING PRE-TRAINED NEURAL NETWORKS HAS BEEN PRESENTED AND EVALUATED ON TWO APPROACHES FOR MULTIPLIERLESS EXECUTION OF DNNS

• We reviewed stochastic rounding of weights for bitwise neural networks and its efficient hardware implementation

• We discussed a method for quantizing pre-trained neural networks without the need for fine-tuning on labeled training data
  - Minimizing the propagated quantization error

• Two approaches for few-bit quantization and multiplierless processing were discussed
  - Logarithmic number representation with arbitrary log-base
  - Mixed-precision log-based quantization (“one-hot”/“two-hot”)
ML-enabled RISC-V

Evaluation of RISC-V core with custom DSP extensions for ML applications
RISC-V IS AN OPEN INSTRUCTION SET ARCHITECTURE WITH A LARGE COMMUNITY

- The ISA definition is open sourced under BSD license
- The RISC-V foundation has a considerable number of commercial partners
- RISC-V may provide the ecosystem for future application specific optimized computing

[10] https://riscv.org/members/
Several customizable cores implementing the RISC-V ISA have been made available in recent years

- Academia (RISC-V Virtual Prototype Uni Bremen, PULP platform ETH Zürich)
- Industry (e.g., through Synopsis ASIP designer, codasip, imperas, SiFive, etc.)

MODEL-BASED DESIGNS ALLOW FAST EVALUATION OF CUSTOMIZED RISC-V ISA & CORES

• To evaluate a RISC-V ISA extension on a given use-case, it is beneficial to rely on a virtual-prototype-approach for
  - faster development
  - easier debugging
  - selecting the appropriate abstraction level

• To simplify custom extensions integration, automatic generation of compiler backends for automatic inference of customized instructions is key and makes it user-friendly

• To ease verification of design, automatic generation of RTL from virtual-prototype is a beneficial feature

By integrating dedicated DSP-extensions, a RISC-V core achieves competitive results, outperforming a commercially available MCU.

- An extension dedicated to ML algorithms has recently been published:
  - Xpulp ISA extension[12]
  - DSP-inspired – zero-overhead loops, fixed-point support, post-increment addressing, etc.

- These extensions can be integrated in optimized libraries through compiler inference or by intrinsics and inline assembly.

- An extended RISC-V core outperforms commercially available MCU with dedicated NN-acceleration library [13].

<table>
<thead>
<tr>
<th>Instruction format</th>
<th>Description</th>
<th>Instruction format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Loop Instructions</td>
<td></td>
<td>Fixed Point Instructions</td>
<td></td>
</tr>
<tr>
<td>lp.starti L, I</td>
<td>Set the HW loop start address</td>
<td>p.add[R, N] rD, rA, rB, I</td>
<td>Addition with round and norm. by 1 bits</td>
</tr>
<tr>
<td>lp.endi L, I</td>
<td>Set the HW loop end address</td>
<td>p.sub[R, N] rD, rA, rB, I</td>
<td>Subtraction with round and norm. by 1 bits</td>
</tr>
<tr>
<td>lp.count I, rA</td>
<td>Set the number of iterations</td>
<td>p.mul [hn] R, rA, rB, I</td>
<td>Multi. with round and norm. by I bits</td>
</tr>
<tr>
<td>lp.setupi L, I, I</td>
<td>HW loop setup with immediate</td>
<td>p.clip I rD, rA, I</td>
<td>Clip the value between $-2^{i-1}$ and $2^{i-1}-1$</td>
</tr>
</tbody>
</table>

Extended Load/Store Instructions

- p.l[b, h, w] rD, rB, rA, [rA][rB] load a value from address (rA + (rB))<sup>c</sup>
- p.lf[b, h, w] rD, rB, rA, [rA][rB] load a value from address rA and increment rA by (rB)<sup>c</sup>
- p.s[b, h, w] rB, [rD][rA] store a value to address (rA + (rD))<sup>c</sup>
- p.sf[b, h, w] rB, [rD][rA] store a value to address rA and increment rA by (rD)<sup>c</sup>

Vectorial Instructions

- p.v.instr [b, h] rD, rA, rB vectorial instruction between two registers<sup>c</sup>
- p.v.instr [b, h] rD, rA, rB vectorial instr. between a register and an immediate<sup>c</sup>

The RISC-V open ISA is a suitable vehicle to integrate and evaluate novel approaches for efficient processing of DNNs such as number representations and quantized processing aspects.
References

[1] Hubara et al., Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1, NIPS 2016


[5] Vogel et al., Efficient Hardware Acceleration for Approximate Inference of Binarized Deep Neural Networks, DASIP 2017


[7] Vogel et al., Self-Supervised Quantization of Pre-Trained Neural Networks for Multiplierless Acceleration, DATE 2019

[8] Vogel et al., Efficient Hardware Acceleration of CNNs Using Logarithmic Data Representation with Arbitrary Log-base, ICCAD 2018

[9] Vogel et al., Bit-Shift-Based Accelerator for CNNs with Scalable Accuracy and Throughput, DSD 2019

[10] https://riscv.org/members/


[12] Gautschi et al., Near-Threshold RISC-V Core With IQE Extensions for Scalable IoT Endpoint Devices, VLSI 2017

**LEMONADE (RESULTS FOR AUTOMATED DESIGN OF ERROR-RESILIENT DNNS)**

- Automated design of error-resilient and hardware-efficient deep neural networks
- Optimizing for robustness / error resiliency wrt. perturbations (bitflips) in activation values
- Using evolutionary multi-objective optimization method LEMONADE\[1\]

**Evaluation:**
- Bit Error Rate determined by fault injection

**Observation:**
- minPQE $\rightarrow$ more robust DNNs than min-max quant.

[2] Vogel et al., *Self-Supervised Quantization of Pre-Trained Neural Networks for Multiplierless Acceleration*, DATE 2019

---

**Graphs:**
- Classification Change Rate vs. Bit Error Rate
- WorstASI Model vs. BestASI Model