HAWIS: Hardware-Aware Automated WIdth Search for Accurate, Energy-Efficient and Robust Binary Neural Network on ReRAM Dot-Product Engine

Qidong Tang, Zhezhi He*, Fangxin Liu, Zongwu Wang,
Yiyuan Zhou, Yinghuan Zhang and Li Jiang*

Shanghai Jiao Tong University

tangqidong@sjtu.edu.cn

December 11 2021

Presenter Information

A second-year Master student in Computer Science at the Advanced Computer Architecture Laboratory of SJTU.

PIM and ReRAM

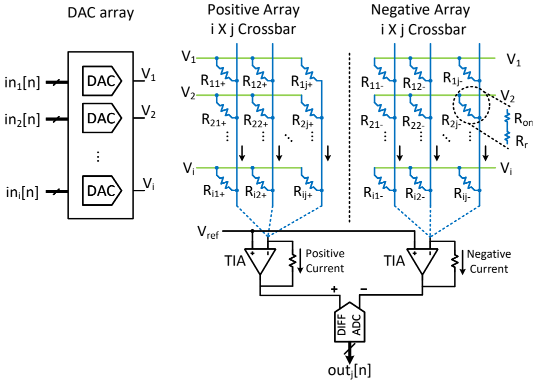


Fig. Hardware implementation of $M \times M$ ReRAM crossbar array pair as an analog dot-product engine.

1. ReRAM represents the weight by dividing the resistance range into multiple intervals.
2. The input is encoded as binary bit-strings $in_i[n]$ for crossbar input with DACs.
3. $$I_k = \sum_{i=1}^{M} \left(\frac{V_i}{R_{ik}^{+}} - \frac{V_i}{R_{ik}^{-}} \right)$$

   The current is transformed into digital calculation results with ADCs.

Why BNN on ReRAM?

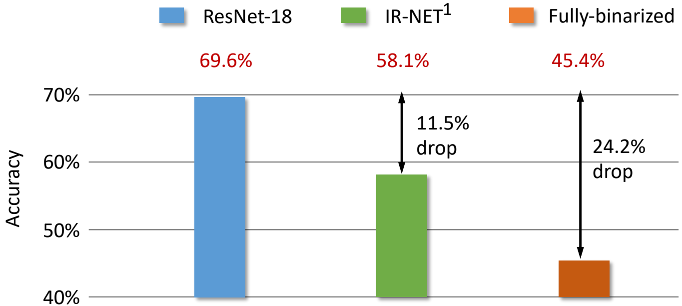
Motivations to deploy BNN (1 bit network) to ReRAM:

1. Simplify the hardware-expensive peripheral circuits (e.g, DAC), which commonly consume a great portion of ($> 50\%$) on-chip area and energy.
2. Minimize the storage footprint and reduce the model size by $32\times$.
3. Superior bit error tolerance[1], which inspires us to make use of this capability to overcome the severe device defects in ReRAM, such as resistance variation and Stuck-At-Fault (SAF).

[1] Adnan Siraj Rakin, Zhezhi He e Deliang Fan. "Bit-flip attack: Crushing neural network with progressive bit search". Em: *ICCV*. 2019.

Challenges to deploy BNN on ReRAM?

1. Drastic accuracy degradation[2](11.5% accuracy drop);
2. Applying binarization to the whole network will further lower the accuracy(24.2% accuracy drop).



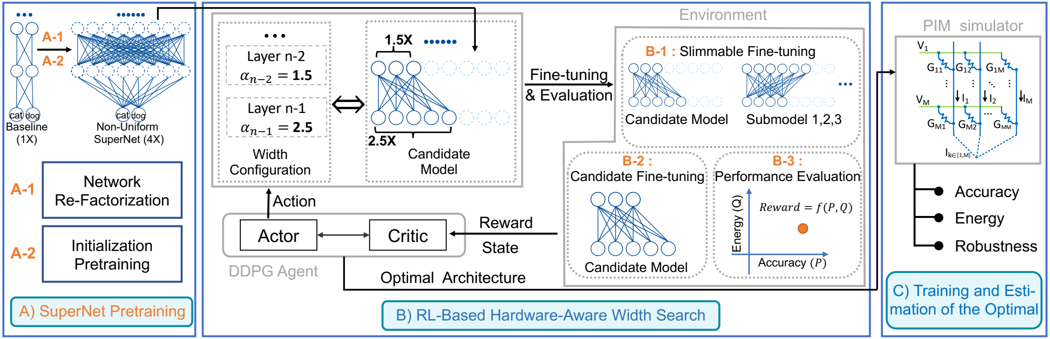
[2] Haotong Qin et al. "Forward and backward information retention for accurate binary neural networks". Em: *CVPR*. 2020.

Our main idea – searching the width of BNN on ReRAM.

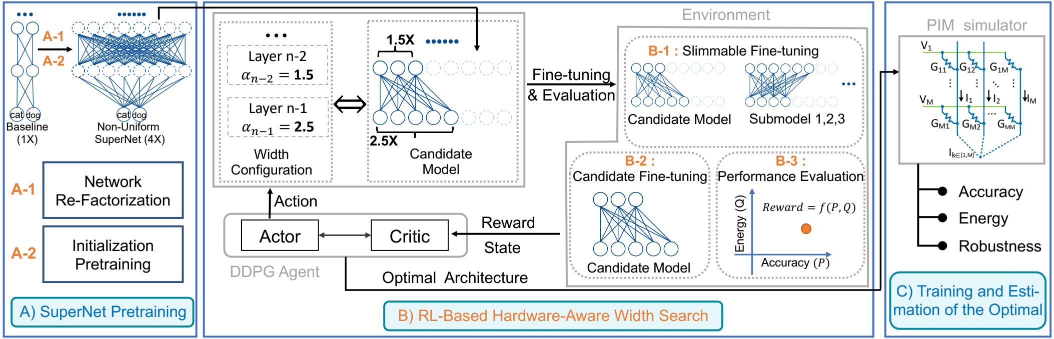
It is effective to widen the quantized network to mitigate the accuracy drop[3, 4].
However, the same expansion ratio across the network leads to model overfitting.
$\Rightarrow$ Thus we utilize reinforcement learning to determine the specific width layer by layer.

Model	**Res-20 CIFAR10**		**Res-32 CIFAR10**		**Res-18 ImageNet**	
	Energy (μJ)	Acc. (%)	Energy (μJ)	Acc. (%)	Energy (mJ)	Acc. (%)
Quan-8bit	1387	92.2	2349	92.9	66.5	69.8
Uniform-BNN 1×	32.7	81.22	50.6	83.91	3.8	51.92
Uniform-BNN 2×	120	88.95	195	90.22	8.2	63.38
Uniform-BNN 3×	238	91.4	393	92.11	15.0	66.57
Uniform-BNN 4×	503	92.17	893	92.49	25.1	68.19
Uniform-BNN 5×	924	92.77	1571	93.00	43.5	69.22
Uniform-BNN 6×	1176	92.78	1984	93.07	-	-

[3] Asit Mishra et al. "WRPN: Wide reduced-precision networks". Em: *ICLR* (2018).
[4] Mingzhu Shen et al. "Searching for accurate binary neural architectures". Em: *ICCV Workshops*. 2019.

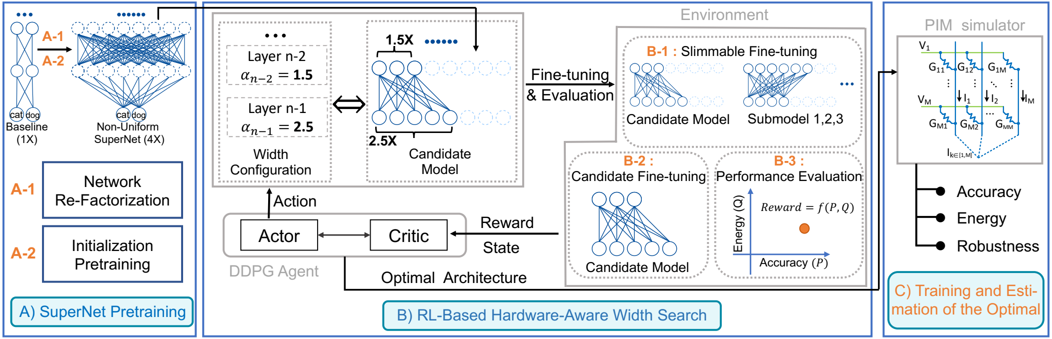


A). Train a binarized super-net.



A). Train a binarized super-net.

B). Leverage reinforcement learning to search for the width layer-by-layer.



A). Train a binarized super-net.

B). Leverage reinforcement learning to search for the width layer-by-layer.

C). Estimate the accuracy, energy consumption and robustness.

Stage-A creates and pretrains a binarized super-net, greatly reducing the search cost.

- **Binarization Function Insertion** to all parametric layers.

Stage-A creates and pretrains a binarized super-net, greatly reducing the search cost.

- **Binarization Function Insertion** to all parametric layers.
- **Topology Modification**: remove the avg-pooling. (45.43% $\rightarrow$ 50.24%)

Stage-A creates and pretrains a binarized super-net, greatly reducing the search cost.

- **Binarization Function Insertion** to all parametric layers.
- **Topology Modification**: remove the avg-pooling. (45.43% → 50.24%)
- **Two-Side Regularization**: $\Omega(w) = \sum_i (|w_i| - w_0)^2$ (50.24% → 51.92%)

Stage-A creates and pretrains a binarized super-net, greatly reducing the search cost.

- **Binarization Function Insertion** to all parametric layers.
- **Topology Modification**: remove the avg-pooling. ($45.43\% \rightarrow 50.24\%$)
- **Two-Side Regularization**: $\Omega(w) = \sum_i (|w_i| - w_0)^2$ ($50.24\% \rightarrow 51.92\%$)
- **Uniform Layer Width Expansion and Pretraining.** Uniformly expand the binarized baseline to create the **super-net**. Leverage the slimmable training technique to pretrain the super-net.

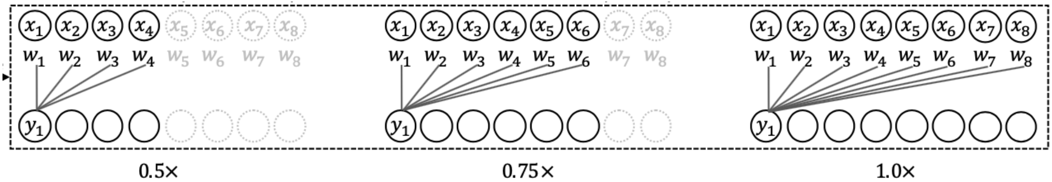
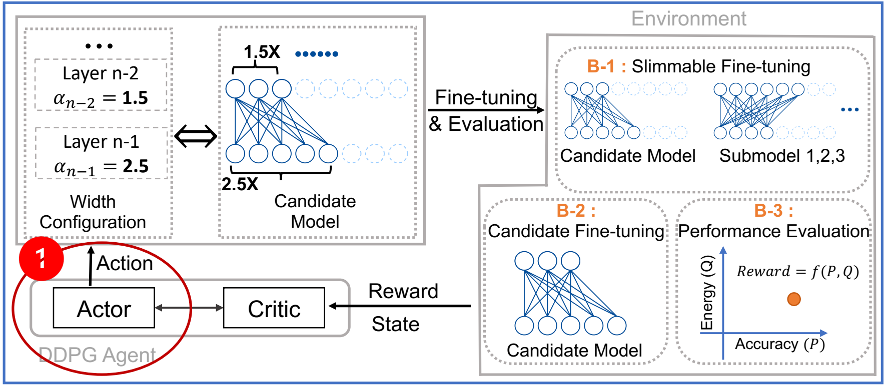


Fig. Slimmable Training Technique.

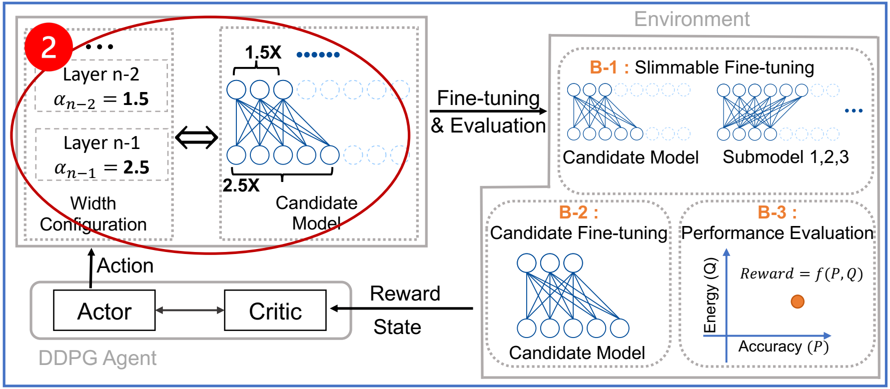
In stage-B, we leverage RL to determine the width in a layer-by-layer manner.

1. The agent takes the state as input and outputs an action (the width configuration).



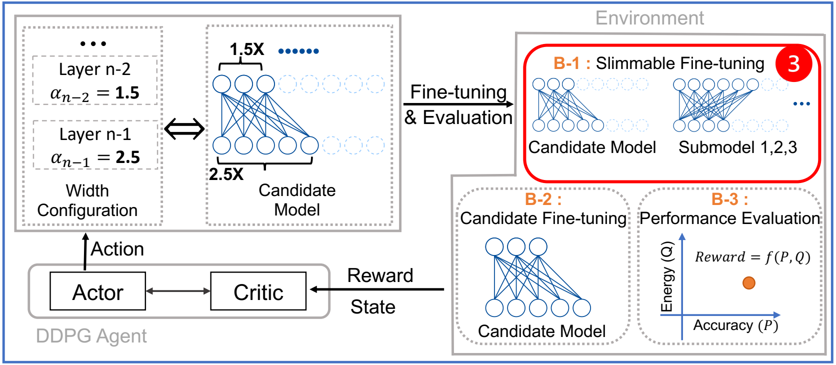
In stage-B, we leverage RL to determine the width in a layer-by-layer manner.

1. The agent takes the state as input and outputs an action (the width configuration).
2. A candidate model w.r.t the current width configuration is generated.



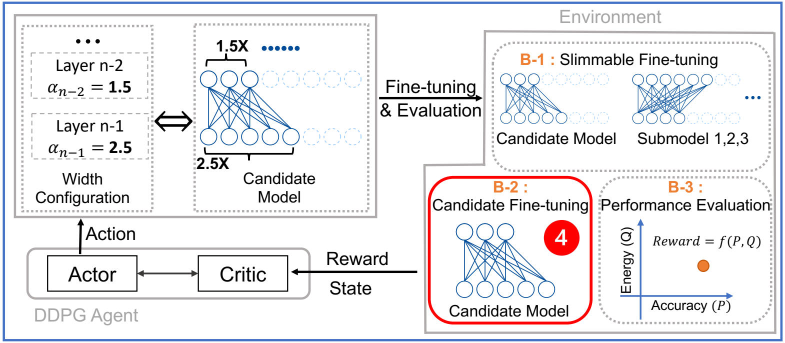
In stage-B, we leverage RL to determine the width in a layer-by-layer manner.

1. The agent takes the state as input and outputs an action (the width configuration).
2. A candidate model w.r.t the current width configuration is generated.
3. Slimmable fine-tuning first updates the super-net (B-1).



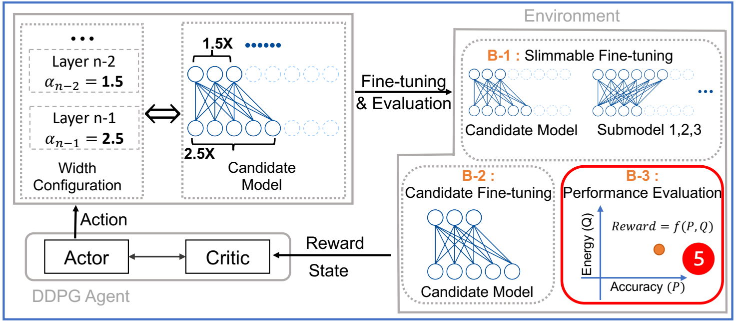
In stage-B, we leverage RL to determine the width in a layer-by-layer manner.

1. The agent takes the state as input and outputs an action (the width configuration).
2. A candidate model w.r.t the current width configuration is generated.
3. Slimmable fine-tuning first updates the super-net (B-1).
4. The candidate is initialized according to the super-net and fine-tuned few epochs (B-2).



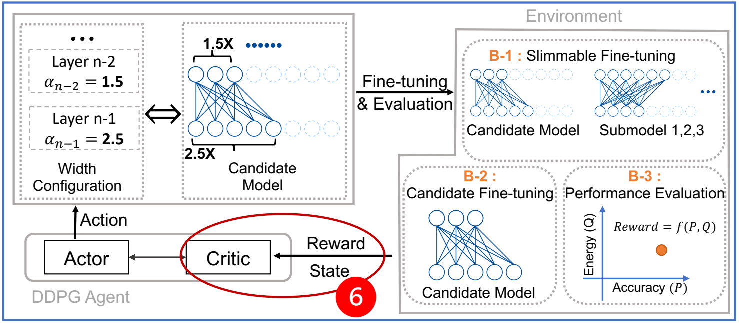
In stage-B, we leverage RL to determine the width in a layer-by-layer manner.

1. The agent takes the state as input and outputs an action (the width configuration).
2. A candidate model w.r.t the current width configuration is generated.
3. Slimmable fine-tuning first updates the super-net (B-1).
4. The candidate is initialized according to the super-net and fine-tuned few epochs (B-2).
5. Performance evaluation estimates the accuracy and energy consumption (B-3).



In stage-B, we leverage RL to determine the width in a layer-by-layer manner.

1. The agent takes the state as input and outputs an action (the width configuration).
2. A candidate model w.r.t the current width configuration is generated.
3. Slimmable fine-tuning first updates the super-net (B-1).
4. The candidate is initialized according to the super-net and fine-tuned few epochs (B-2).
5. Performance evaluation estimates the accuracy and energy consumption (B-3).
6. The reward is returned to update the agent and generate actions in the successive episode.



1. ***Problem Formulation:***
   $\mathcal{A}_{\mathrm{b}}$ — binarized baseline;
   $\mathcal{B}_{\mathrm{b}}$ — sampled sub-net during the search.

$$\mathcal{R} = -\overbrace{\mathrm{Error}\left(\mathcal{B}_{\mathrm{b}}^{*}(\hat{\boldsymbol{\theta}}), \mathrm{X}_{\mathrm{eval}}\right)}^{\textit{Accuracy} \quad \textit{Gap}} \cdot \log \overbrace{\frac{Q\left(\mathcal{B}_{\mathrm{b}}^{*}(\hat{\boldsymbol{\theta}})\right)/\lambda}{Q\left(\mathcal{A}_{\mathrm{b}}\right)}}^{\textit{Energy} \quad \textit{Consumption}} \tag{1}$$

2. ***State Space:***

$$s_l = \left(l, l_{\mathrm{s}}, c_{\mathrm{in}}, c_{\mathrm{out}}, n_{\mathrm{ker}}, n_{\mathrm{str}}, n_{\mathrm{param}}, n_{\mathrm{fmap}}, a_{l-1}, c_{l-1}\right)$$

l, l_{s}	—	layer/block index;
$c_{\mathrm{in}}, c_{\mathrm{out}}$	—	#(input/output channels);
$n_{\mathrm{ker}}, n_{\mathrm{str}}$	—	kernel/stride size;
n_{param}	—	#(parameter);
n_{fmap}	—	#(feature map);
a_{l-1}	—	action of the previous layer;
c_{l-1}	—	expanded channel number of the previous layer.

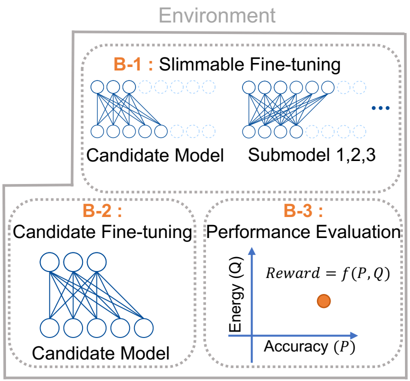
3. ***Action Space:***

   a_l — action for l-th layer;
   r_l — expansion ratio for l-th layer;
   c_l — the actual channel number of l-th layer.

$$
\begin{aligned}
r_l &= a_l\left(r_{\max} - r_{\min}\right) + r_{\min} \\
c_l &= \operatorname{round}\left(c_{\text{out}} \cdot r_l / d\right) \cdot d
\end{aligned}
$$

4. ***Environment:***

- B-1) : Slimmable training technique updates the super-net for 1 epoch on the training data.
- B-2): Customized fine-tuning for the candidate model for a few epochs.
- B-3): Estimate the accuracy and energy consumption of the candidate model on the evaluation data.



Training and Estimation of the Optimal Model

Train from scratch, estimate the final accuracy, energy consumption and robustness under device defects.

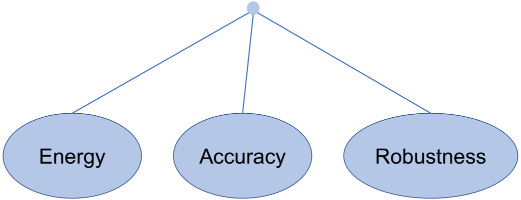


Table. Comparison of High Bit-width, Uniformly Widened Binarized (U-) and HAWIS networks.

Model	Res-20 CIFAR10		Res-32 CIFAR10		Res-18 ImageNet	
	Energy (μJ)	Acc. (%)	Energy (μJ)	Acc. (%)	Energy (mJ)	Acc. (%)
FP	-	92.1	-	92.8	-	69.6
Quan-8bit	1387	92.2	2349	92.9	66.5	69.8
U-1$\times$	32.7	81.22	50.6	83.91	3.8	51.92
U-2$\times$	120	88.95	195	90.22	8.2	63.38
U-3$\times$	238	91.4	393	92.11	15.0	66.57
U-4$\times$	503	92.17	893	92.49	25.1	68.19
U-5$\times$	924	92.77	1571	93.00	43.5	69.22
U-6$\times$	1176	92.78	1984	93.07	-	-
HAWIS-A	368	92.42	949	92.91	21.3	68.21
HAWIS-B	849	93.13	1045	93.18	29.4	69.29

1. HAWIS models achieve better overall performance, which consume less energy to reach similar accuracy of uniformly widened BNNs.
2. On CIFAR-10, HAWIS-A models reach the accuracy of Quan-8bit models. On ImageNet, the accuracy of HAWIS-B is 0.5% lower than that of Quan-8bit model.

Comparison Against State-of-the-Art Efficient Models

Table. Performance and Complexity Comparison on CIFAR-10.

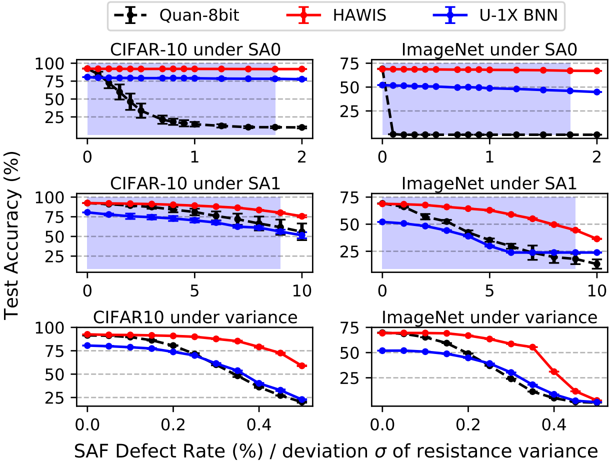
Arch	Precision (W/A)	BiOps ($\times 10^6$)	FLOPs ($\times 10^6$)	Search Cost (GPU-days)	Top-1 (%)
ResNet-20 [**resnet**]	8/8	0	41	-	92.2
Bi-Real-18 [**bi-real**]	1/1	561	11	-	91.2
BARS [**bars**]	1/1	1048	2	-	92.98
BNAS [**bnas_zeroise**]	1/1	670	3	0.42	92.7
BATS [**bulat2020bats**]	1/1	410	30	0.25	93.7
HAWIS	**1/1**	**1100**	**0**	**1.25**	**93.13**

Table. Performance and Complexity Comparison on ImageNet.

Arch	Precision (W/A)	BiOps ($\times 10^9$)	FLOPs ($\times 10^8$)	Search Cost (GPU-days)	Top-1 (%)
Resnet-18 [**resnet**]	8/8	0	18.2	-	69.8
Bi-Real-18 [**bi-real**]	1/1	1.68	1.38	-	56.4
Bi-Real-34 [**bi-real**]	1/1	3.53	1.39	-	62.2
MeliusNet-42 [**melius**]	1/1	9.69	1.74	-	69.2
FracBNN [**FracBNN**]	1/1.4	7.30.	0.01	-	71.8
BARS [**bars**]	1/1	2.59	2.54	-	60.3
BNAS [**bnas_zeroise**]	1/1	15.30	4.10	0.42	63.5
BATS [**bulat2020bats**]	1/1	2.16	1.21	0.25	66.1
Res18-Auto [4]	1/1	19.40	3.55	60	69.7
HAWIS	**1/1**	**37.8**	**0**	**16**	**69.3**

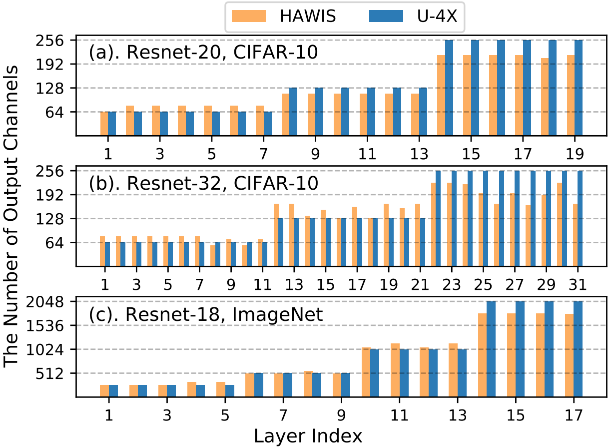
1. HAWIS on CIFAR-10, with fully binarized layers, outperforms all above efficient models except BATS (many full-precision operations).
2. On ImageNet, HAWIS outperforms most manually designed BNNS and Binary NAS methods which still own a large part of FLOPs.

Robustness Under Device Defects



1. Quan-8bit networks are susceptible to SA0 defects, while binarized models keep stable accuracy under SA0 defects.
2. U-1× BNN is more robust than Quan-8bit models under SA1 and resistance variation, while HAWIS further improves the robustness of the binary baseline.

Analysis of the Searched Architecture



1. HAWIS architectures commonly possess more channels in the front layers and fewer channels in the tail layers.
2. HAWIS has a bottleneck-like structure in ResNet-32.(narrow width for 8/18/28 and larger width for 9/19/29-th layer).
3. The selected channel numbers are energy-efficient(full utilization).

Thanks for your listening!