AI ROBOTICS KR

SqueezeNext: Hardware-Aware Neural Network Design

Neural Network Quantization & Compact Network Design Study, AI Robotics KR

Constant (Sang-Soo) Park

https://constantpark.github.io/

2019-10-06

Contents

- Introduction
- Background
- Contributions
- Design: SqeeezeNext/Squeezelerator
- Evaluation
- Discussion

Introduction

- **General trend: Designing a larger and deeper models to get better accuracy[1]**
  - Without considering memory and power budget
  - **Improvement of transistor is slow, hard to meet computational requirement in batter-driven device**

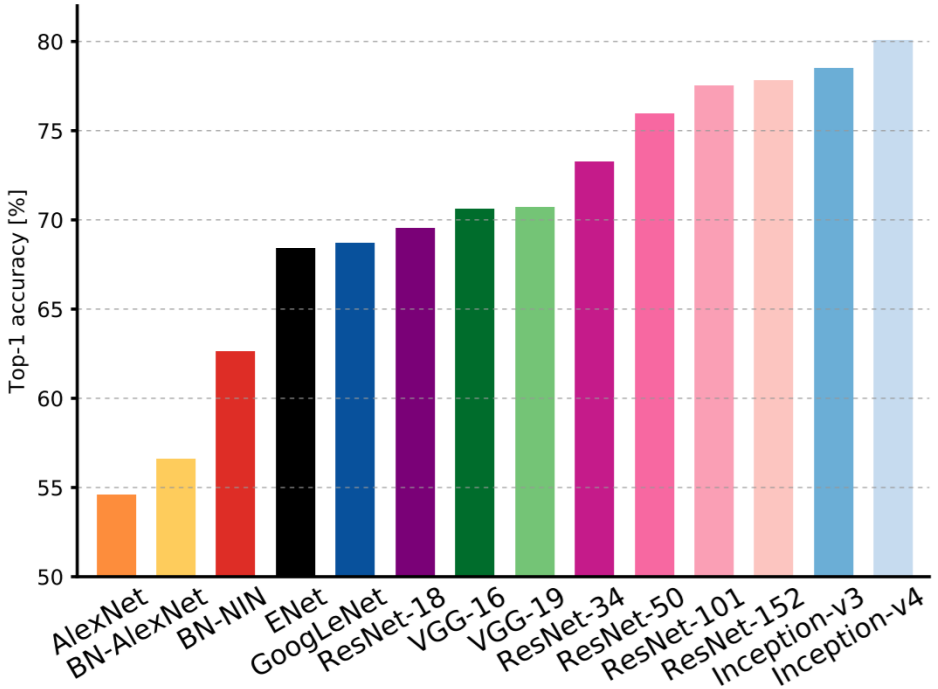


Figure 1: **Top1 *vs.* network.** Si

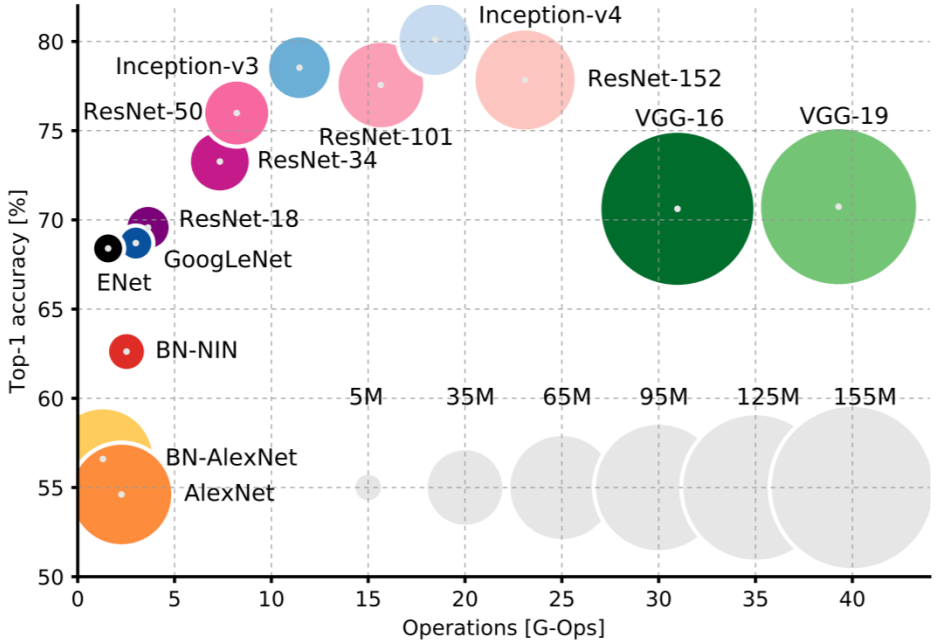
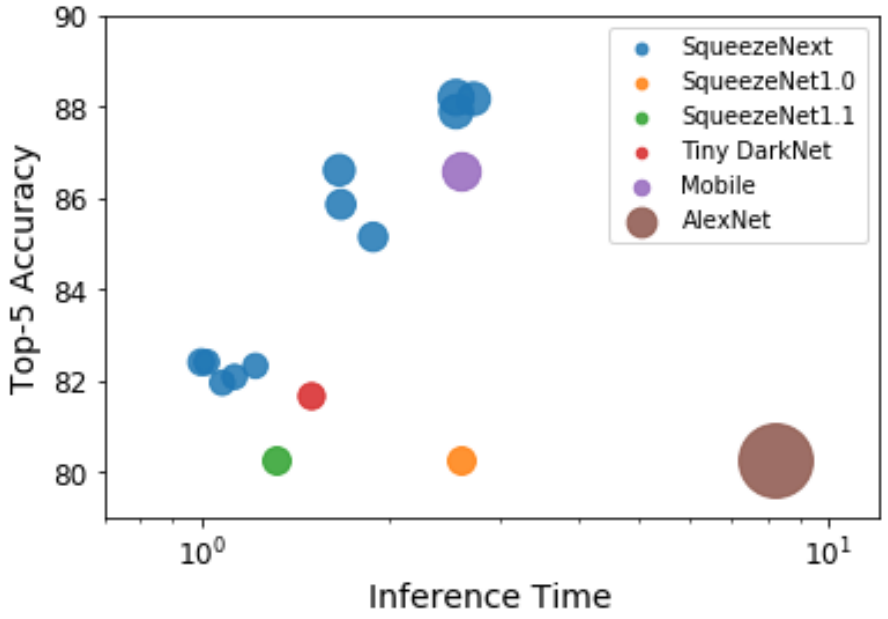


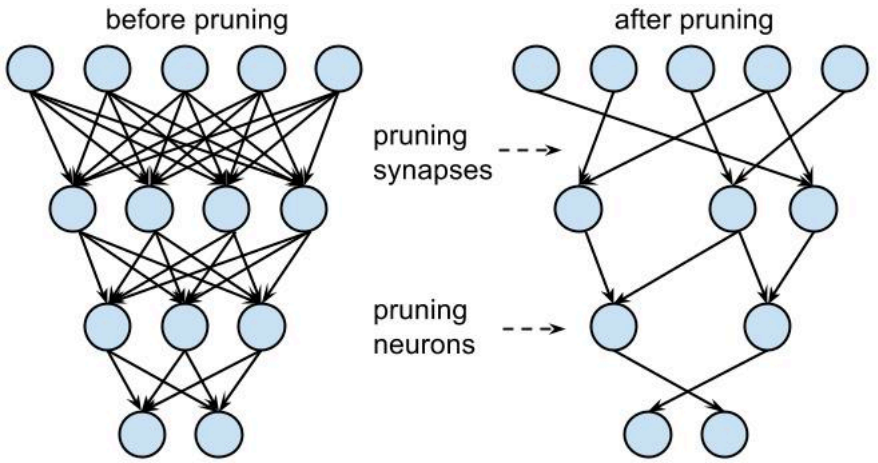
Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.**

[1] AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS, ICLR, 2017

Background: Lightweight NNA

- **Designing low memory usage and MAC operations**
  - The lower MAC operation and memory usage, The lower cost can be possible
  - Lightweight neural network: SqueezeNet, TinyDarknet, ShuffleNet, CondenseNet





Contributions

- **Three changes compared to SqueezeNet**
  - More aggressive channel reduction by incorporating a two-stage squeeze module
  - Using separable 3×3 convolution and removing 1×1 branch after squeeze module
  - Element-wise addition skip connection similar to that of ResNet architecture
  - **Squeezelerator: Multi-processor embedded system[2]**

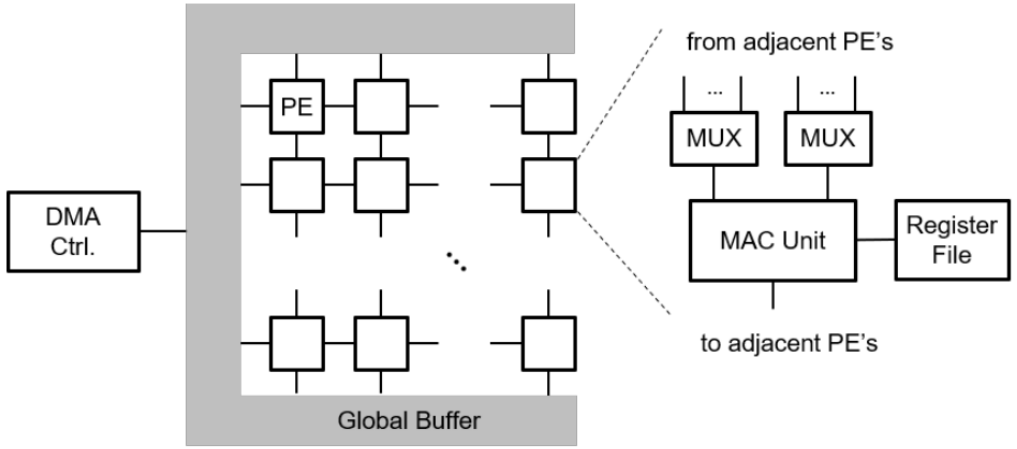


Figure 5: *Block diagram of the neural network accelerator used as the reference hardware for inference speed and energy estimation of various neural networks.*

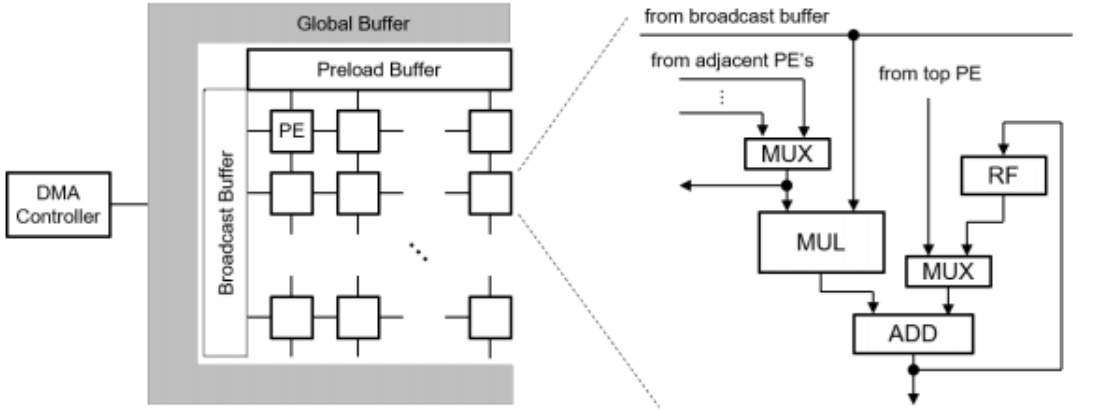


Figure 2: The block diagram of Squeezelerator (left) and PE (right)

[2] Co-Design of Deep Neural Nets and Neural Net Accelerators for Embedded Vision Applications, DAC, 2017.

SqueezeNext Design: Low Rank Filters

- ***K×K* convolutions into two separable convolutions of size *1×K* and *K×1***
  - CP[3] or Tucker decomposition[4]
  - Reducing the number of parameters from K^2 to $2K$

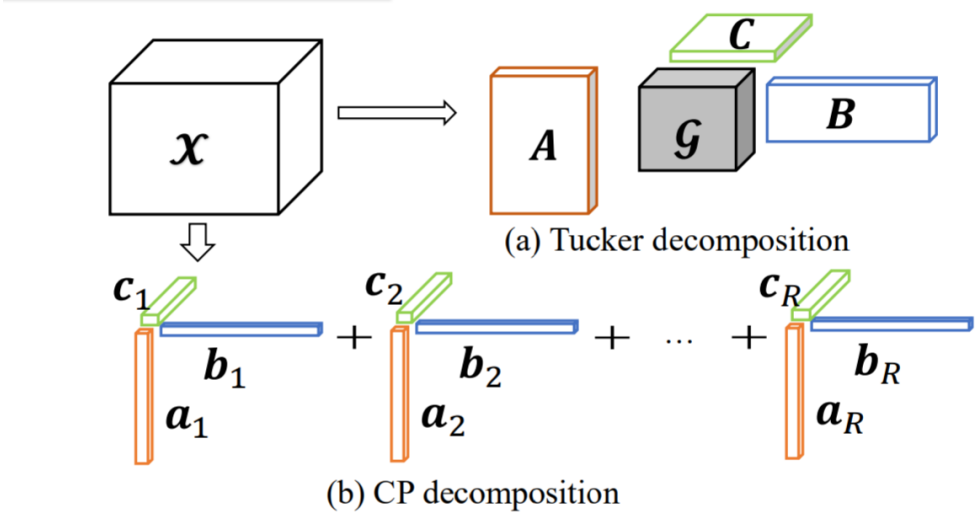


Fig. 1. The illustration of (a) Tucker decomposition and (b) CP factorization of an $n_1 \times n_2 \times n_3$ tensor.

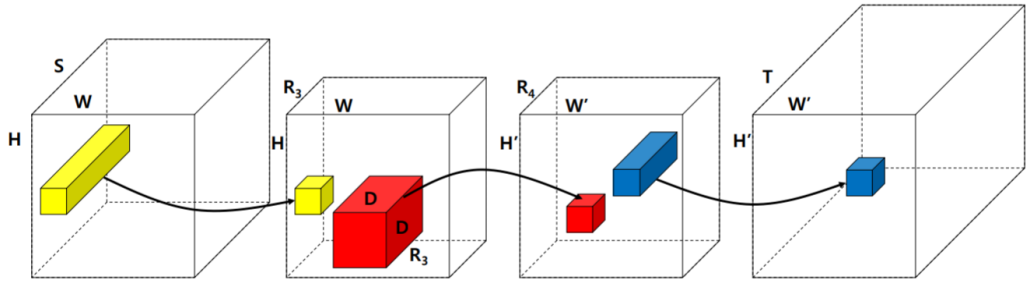


Figure 3: Tucker-2 decompositions for speeding-up a convolution. Each transparent box corresponds to 3-way tensor $\mathcal{X}$, $\mathcal{Z}$, $\mathcal{Z}'$, and $\mathcal{Y}$ in (3-5), with two frontal sides corresponding to spatial dimensions. Arrows represent linear mappings and illustrate how scalar values on the right are computed. Yellow tube, red box, and blue tube correspond to 1×1, $D \times D$, and 1×1 convolution in (3), (4), and (5) respectively.

[3] A novel nonconvex approach to recover the low-tubal-rank tensor data: when t-SVD meets PSSV, arxiv, 2017.
[4] COMPRESSION OF DEEP CONVOLUTIONAL NEURAL NETWORKS FOR FAST AND LOW POWER MOBILE APPLICATIONS, ICLR, 2015.

SqueezeNext Design: Bottleneck Module/FC

AI ROBOTICS KR

- **Bottleneck modules (Two stage squeeze layer)**
  - Reducing the channel size by a factor of 2, followed by two separable convolutions
  - Final 1×1 expansion module, reducing the number of output channels for the separable convolutions
- **Final bottleneck layer to reduce the input channel size to the last fully layer**
  - In AlexNet, FC accounting for 96% of the total memory size

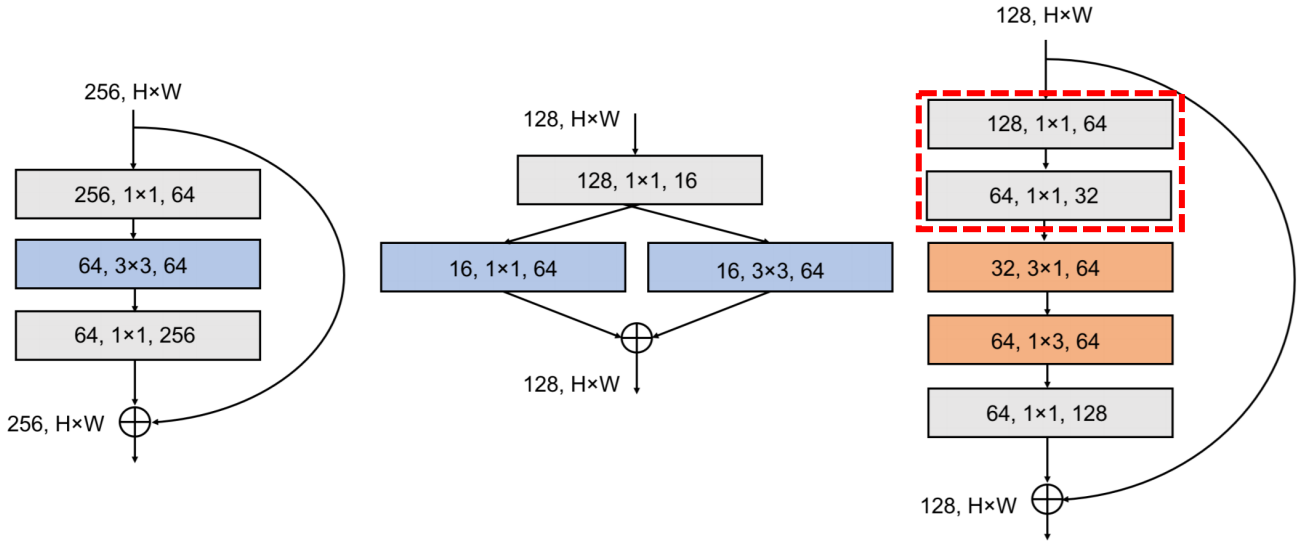


Figure 1: *Illustration of a ResNet block on the left, a SqueezeNet block in the middle, and a SqueezeNext (SqNxt) block on the right. SqueezeNext uses a two-stage bottleneck module to reduce the number of input channels to the* 3×3 *convolution. The latter is further decomposed into separable convolutions to further reduce the number of parameters (orange parts), followed by a* 1×1 *expansion module.*

SqueezeNext Architecture

- **Convolution in SqueezeNext**
  - *1×1* (64) -> *1×1* (32) -> *3×1* (64) -> *1×3* (64) -> *1×1* (128)

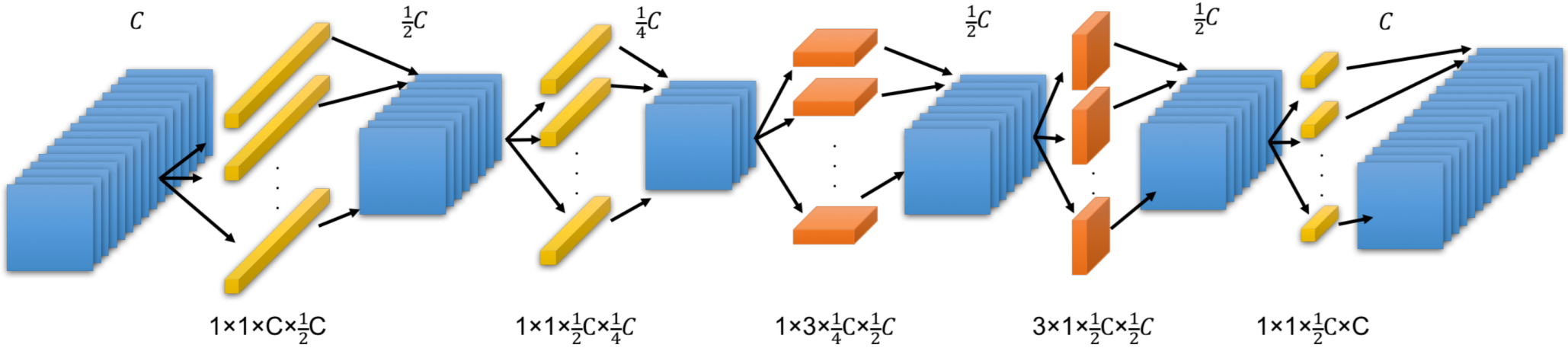


Figure 2: *Illustration of a SqueezeNext block. An input with C channels is passed through a two stage bottleneck module. Each bottleneck module consists of 1×1 convolutions reducing the input channel size by a factor of 2. The output is then passed through a separable 3×3 convolution. The order of 1×3 and 3×1 convolutions is changed throughout the network. The output from the separable convolution is finally passed through an expansion module to match the skip connection's channels (the skip connection is not shown here).*

SqueezeNext Architecture

AI ROBOTICS KR

- **Overall architecture of SqueezeNext**
  - Conv (6) – Conv (6) – Conv (8) – Conv (1)

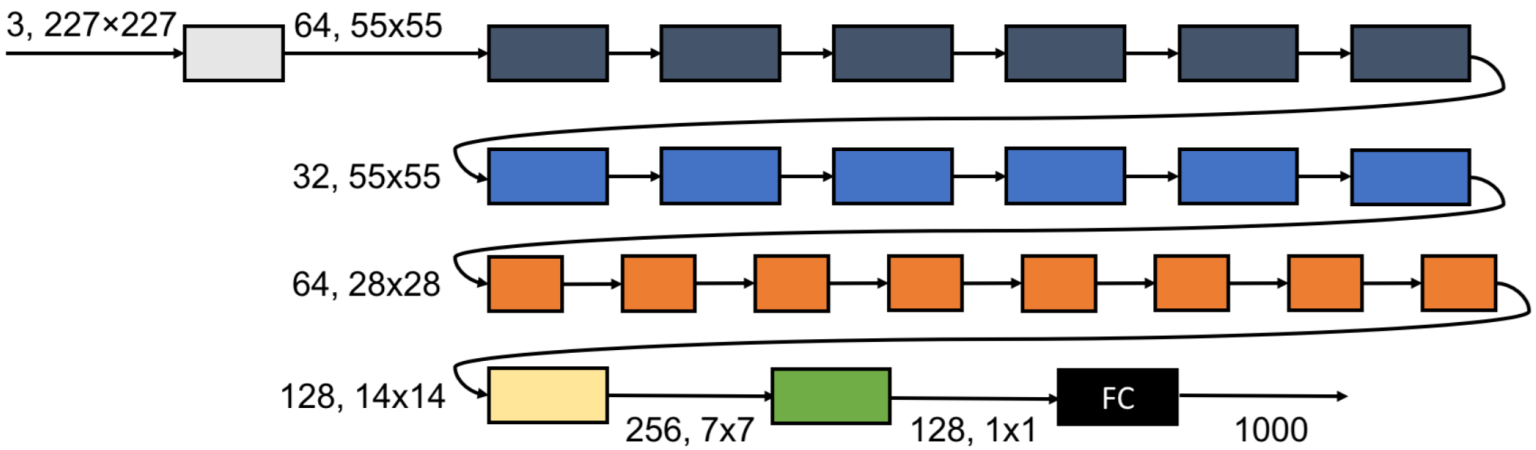
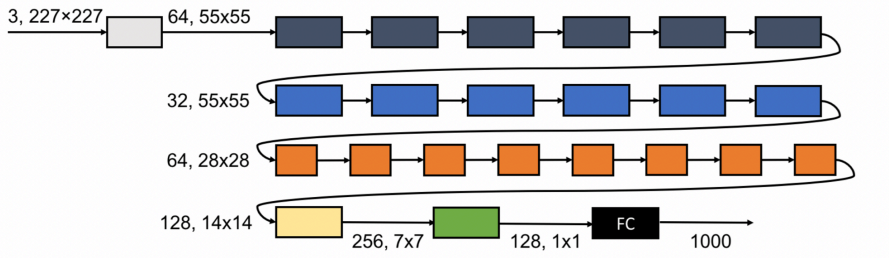
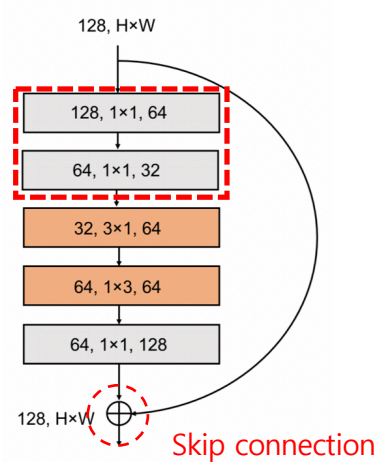


Figure 3: *Illustration of block arrangement in 1.0-SqNxt-23. Each color change corresponds to a change in input feature map's resolution. The number of blocks after the first convolution/pooling layer is* $Depth = [6, 6, 8, 1]$*, where the last number refers to the yellow box. This block is followed by a bottleneck module with average pooling to reduce the channel size and spatial resolution (green box), followed by a fully connected layer (black box). In optimized variations of the baseline, we change this depth distribution by decreasing the number of blocks in early stages (dark blue), and instead assign more blocks to later stages (Fig. 9). This increases hardware performance as early layers have poor compute efficiency.*

SqueezeNext Architecture





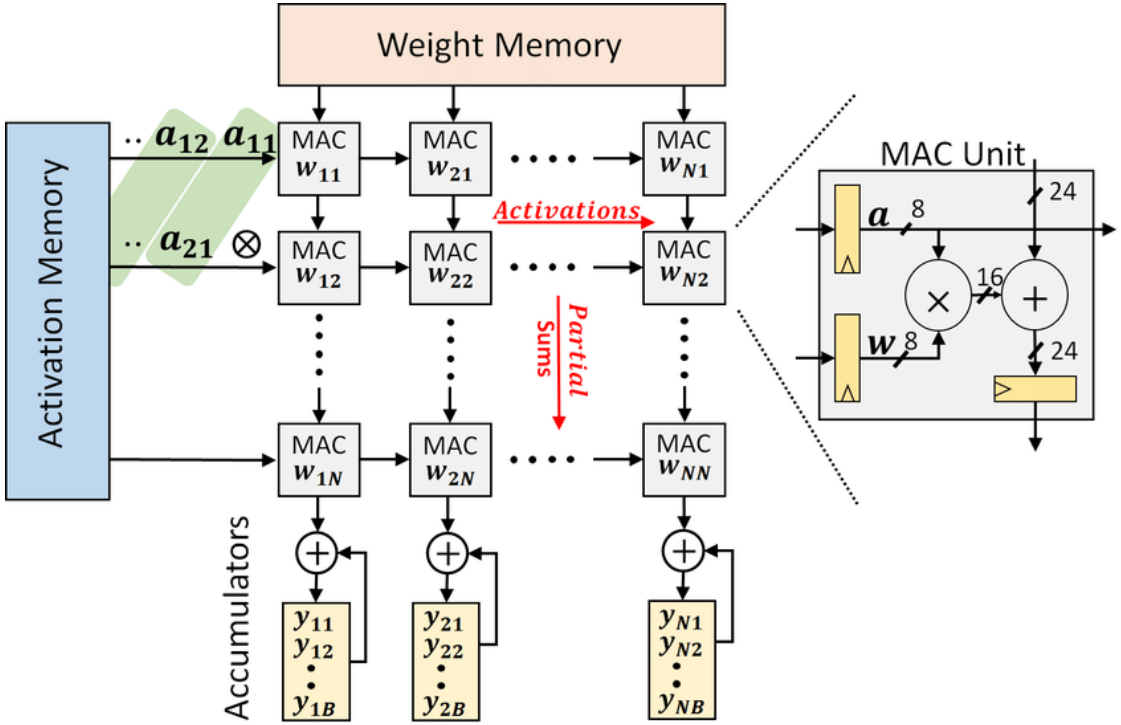
Skip connection

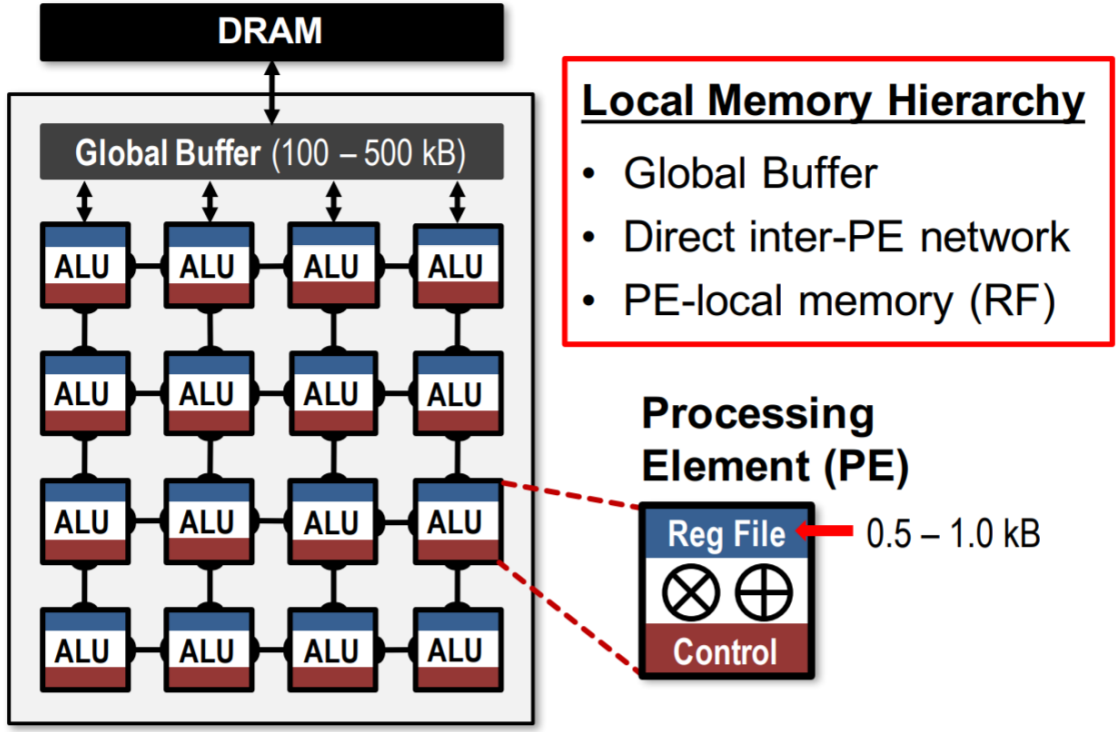
	Name	Type	W_i	H_i	C_i	K_w	K_h	W_o	H_o	C_o	S	P_w	P_h	Repeat	Params	MAC	Act. Size
6	conv1	Conv	227	227	3	7	7	111	111	64	2	0	0	1	9.5	116,705	3080
	conv2	Conv	55	55	64	1	1	55	55	32	1	0	0	1	2.1	6,292	378
	conv3	Conv	55	55	64	1	1	55	55	16	1	0	0	1	1.0	3,146	189
	conv4	Conv	55	55	16	1	1	55	55	8	1	0	0	1	0.1	411	95
	conv5	Conv	55	55	8	1	3	55	55	16	1	0	1	1	0.4	1,210	189
	conv6	Conv	55	55	16	3	1	55	55	16	1	1	0	1	0.8	2,372	189
	conv7	Conv	55	55	16	1	1	55	55	32	1	0	0	1	0.5	1,646	378
	conv8	Conv	55	55	32	1	1	55	55	16	1	0	0	5	2.6	7,986	189
	conv9	Conv	55	55	16	1	1	55	55	8	1	0	0	5	0.7	2,057	95
	conv10	Conv	55	55	8	3	1	55	55	16	1	1	0	5	2.0	6,050	189
	conv11	Conv	55	55	16	1	3	55	55	16	1	0	1	5	3.9	11,858	89
	conv12	Conv	55	55	16	1	1	55	55	32	1	0	0	5	2.7	8,228	378
6	conv33	Conv	55	55	32	1	1	28	28	64	2	0	0	1	2.1	1,656	196
	conv34	Conv	55	55	32	1	1	28	28	32	2	0	0	1	1.1	828	98
	conv35	Conv	28	28	32	1	1	28	28	16	1	0	0	1	0.5	414	49
	conv36	Conv	28	28	16	1	3	28	28	32	1	0	1	1	1.6	1,229	98
	conv37	Conv	28	28	32	3	1	28	28	32	1	1	0	1	3.1	2,434	98
	conv38	Conv	28	28	32	1	1	28	28	64	1	0	0	1	2.1	1,656	196
	conv39	Conv	28	28	64	1	1	28	28	32	1	0	0	5	10.4	8,154	98
	conv40	Conv	28	28	32	1	1	28	28	16	1	0	0	5	2.6	2,070	49
	conv41	Conv	28	28	16	3	1	28	28	32	1	1	0	5	7.8	6,147	98
	conv42	Conv	28	28	32	1	3	28	28	32	1	0	1	5	15.5	12,168	98
	conv43	Conv	28	28	32	1	1	28	28	64	1	0	0	5	10.6	8,279	196
8	conv64	Conv	28	28	64	1	1	14	14	128	2	0	0	1	8.3	1,631	98
	conv65	Conv	28	28	64	1	1	14	14	64	2	0	0	1	4.2	815	49
	conv66	Conv	14	14	64	1	1	14	14	32	1	0	0	1	2.1	408	25
	conv67	Conv	14	14	32	1	3	14	14	64	1	0	1	1	6.2	1,217	49
	conv68	Conv	14	14	64	3	1	14	14	64	1	1	0	1	12.4	2,421	49
	conv69	Conv	14	14	64	1	1	14	14	128	1	0	0	1	8.3	1,631	98
	conv70	Conv	14	14	128	1	1	14	14	64	1	0	0	7	57.8	11,327	49
	conv71	Conv	14	14	64	1	1	14	14	32	1	0	0	7	14.6	2,854	25
	conv72	Conv	14	14	32	3	1	14	14	64	1	1	0	7	43.5	8,517	49
	conv73	Conv	14	14	64	1	3	14	14	64	1	0	1	7	86.5	16,947	49
	conv74	Conv	14	14	64	1	1	14	14	128	1	0	0	7	58.2	11,415	98
1	conv105	Conv	14	14	128	1	1	7	7	256	2	0	0	1	33.0	1,618	49
	conv106	Conv	14	14	128	1	1	7	7	128	2	0	0	1	16.5	809	25
	conv107	Conv	7	7	128	1	1	7	7	64	1	0	0	1	8.3	405	12
	conv108	Conv	7	7	64	1	3	7	7	128	1	0	1	1	24.7	1,210	25
	conv109	Conv	7	7	128	3	1	7	7	128	1	1	0	1	49.3	2,415	25
	conv110	Conv	7	7	128	1	1	7	7	256	1	0	0	1	33.0	1,618	49
	conv111	Conv	7	7	256	1	1	7	7	128	1	0	0	1	32.9	1,612	25
	fc1000	FC	1	1	128	1	1	1	1	1000	1	0	0	1	129.0	129	4

Squeezelerator Design: Architecture

AI ROBOTICS KR

- **Systolic array-based Architecture**
  - Processing Element (PE): including multiply and accumulate (MAC) operator, registers
  - Employing in TPU, Eyeriss, Chain-NN, etc.



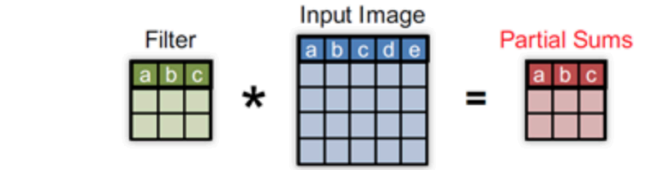


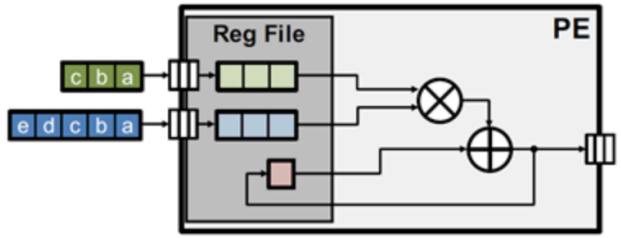
[5] In-Datacenter Performance Analysis of a Tensor Processing Unit, ISCA, 2017.
[6] Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks, ISCA, 2016.

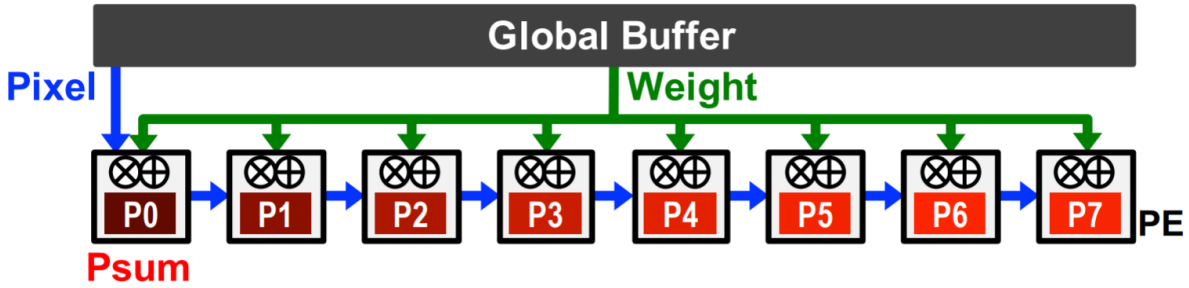
Squeezelerator Design: Dataflow

AI ROBOTICS KR

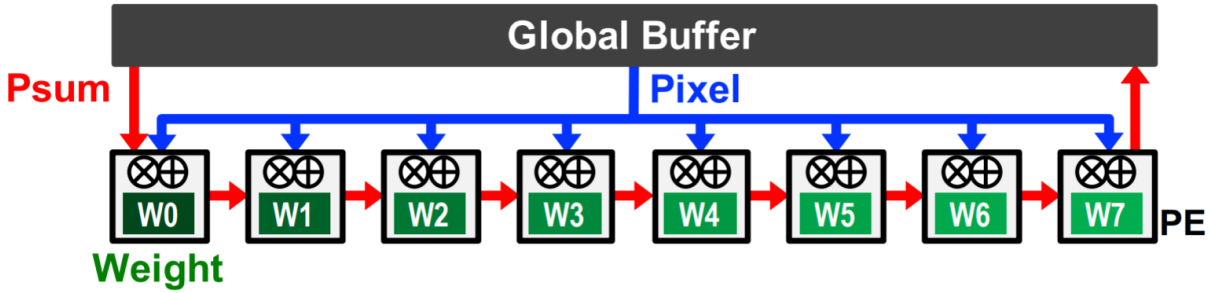
- **Output stationary**
  - Accumulation of each output feature map stays stationary in PEs
  - Minimize **partial sums** R/W energy consumption
- **Weight stationary**
  - Minimize weight read energy consumption
  - Maximize convolution filter reuse
  - Computation between various pixel and one weight







Output stationary



Weight stationary

Squeezelerator Design: Example

- **Stationary example**
  - $K_w \times K_h$ convolution: Multiple loops (MCMK)[7]
  - *x* and *y* loops: WS
  - *c*, *i*, and *j* loops: OS

Algorithm 1: Execution flow for computing a $K_w \times K_h$ convolution kernel

input : Input feature map, I, convolution parameters W
output: Output feature map, O

```
for k ← 0 to C_o do ;                    // Output Channels
 for y ← 0 to H do ;                     // H: Height
  for x ← 0 to W do ;                    // W: Width
   O[k][y][x] = 0;
   for c ← 0 to C_i do ;                 // Input Channels
    for j ← 0 to K_h do ;                // Filter Size
     for i ← 0 to K_w do
      O[k][y][x] += I[c][y+j][x+i] * W[k][c][j][i]
```

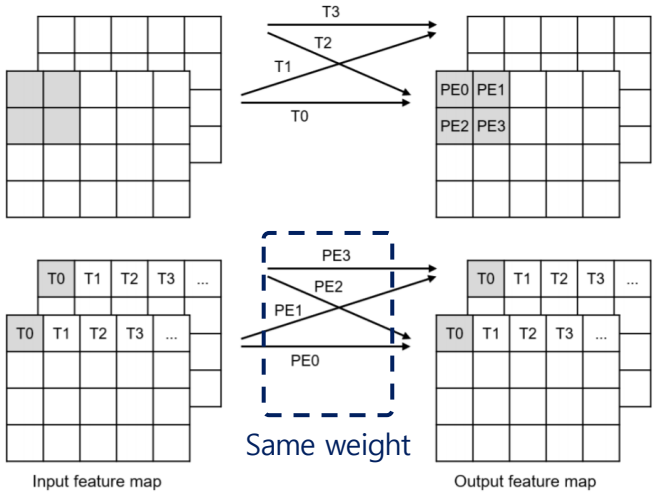


Figure 4: *Conceptual diagram of two data flows used in the experiment: Output Stationary (top) and Weight Stationary (bottom). A 2 × 2 PE array performs a 1 × 1 convolution on a 5x4x2 input (left) generating a 5x4x2 output (right). Here Ti denotes the ith cycle. (Top) In T0, and T1 cycles of OS data flow, the shaded area on left is read and convolved with different filter weights and the results are stored in the corresponding output pixels. Then in T2 and T3 cycles the data from the second input channel are read and similar operation is performed to accumulate partial sums. (bottom) in WS the input pixel is first broadcast to the PEs. In the first cycle, PE0 and PE1 apply different convolutions to the first pixel of first input channel and accumulate the results from PE2 and PE3, respectively, and store the results to the corresponding output pixel. In next cycles other input pixels are read and the same operation is performed.*

[7] Optimal DNN Primitive Selection with Partitioned Boolean quadratic Programming, CGO, 2019.

Squeezelerator Design: HW simulation

AI ROBOTICS KR

- **16×16 or 8×8 PE array**
  - 128KB or 32KB global buffer, DMA
  - 16bit integer MAC unit
  - Assume 40% weight sparsity
  - **Performance is estimated by # of clock cycles to compute each layer**

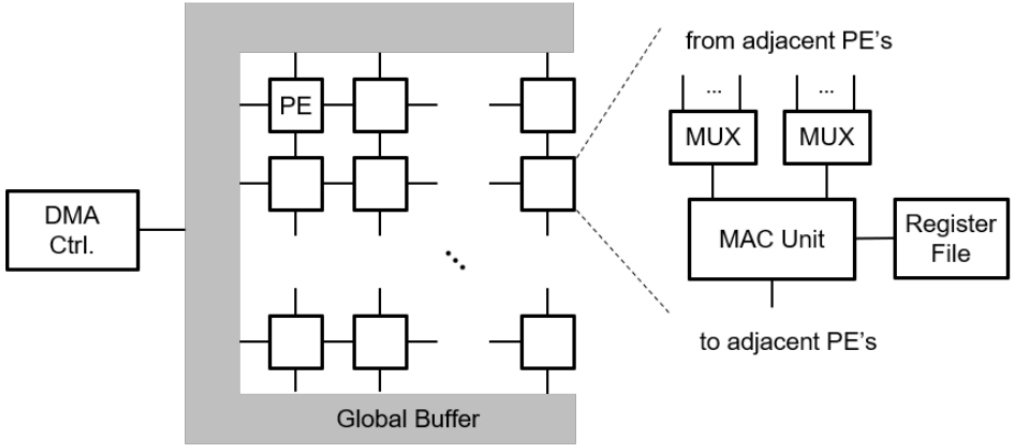


Figure 5: *Block diagram of the neural network accelerator used as the reference hardware for inference speed and energy estimation of various neural networks.*

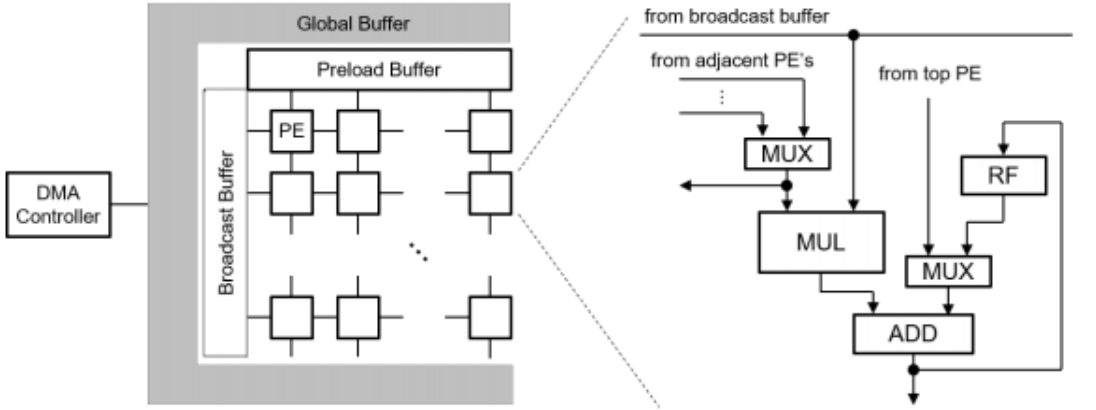


Figure 2: The block diagram of Squeezelerator (left) and PE (right)

Evaluation: SqueezeNext

AI ROBOTICS KR

- **23 module architecture outperforms AlexNet's performance**
  - **W-SqNxt-N-IDA** (W: Width, N:#of module, IDA: Iterative Deep Aggregation)
  - **1.0SqNxt-23**: 2% margin with 84x smaller parameter
  - **1.0-G-SqNxt-23:** Group convolution with a group size of two (112x smaller parameter)
  - **2.0-SqNxt-44** (Twice width and 44 modules): 32x less parameter that of VGG-19

Model	Top-1	Top-5	# Params	Comp.
AlexNet	57.10	80.30	60.9M	1×
SqueezeNet	57.50	80.30	1.2M	51×
1.0-SqNxt-23	58.98	82.33	0.72M	84×
1.0-G-SqNxt-23	**56.88**	**80.83**	**0.54M**	**112×**
1.0-SqNxt-23-IDA	60.35	83.56	0.9M	68×
1.0-SqNxt-34	61.39	84.31	1.0M	61×
1.0-SqNxt-34-IDA	62.56	84.93	1.3	47×
1.0-SqNxt-44	**62.64**	**85.15**	**1.2M**	51×
1.0-SqNxt-44-IDA	63.75	85.97	1.5M	41×

Model	Top-1	Top-5	Params
1.5-SqNxt-23	63.52	85.15	1.4M
1.5-SqNxt-34	65.99	87.40	2.1M
1.5-SqNxt-44	67.27	88.15	2.6M
VGG-19	**68.50**	**88.50**	**138M**
2.0-SqNxt-23	67.18	88.17	2.4M
2.0-SqNxt-34	68.46	88.78	3.8M
2.0-SqNxt-44	**69.08**	**89.36**	**4.4M**
MobileNet	67.50 (70.9)	86.59 (89.9)	4.2M
2.0-SqNxt-23v5	67.44	**88.20**	**3.2M**

Evaluation: Squeezelerator

AI ROBOTICS KR

- **SqueezeNext running on hardware**
  - Acceleration efficacy (# of computation / cycles)
  - **Large filter size applied on a large input feature map (**v1: 7×7 conv, v2: 5×5 conv)
  - Base model (6-6-8-1), v3/v4 (4-8-8-1/2-10-8-1), v5(2-4-14-1)

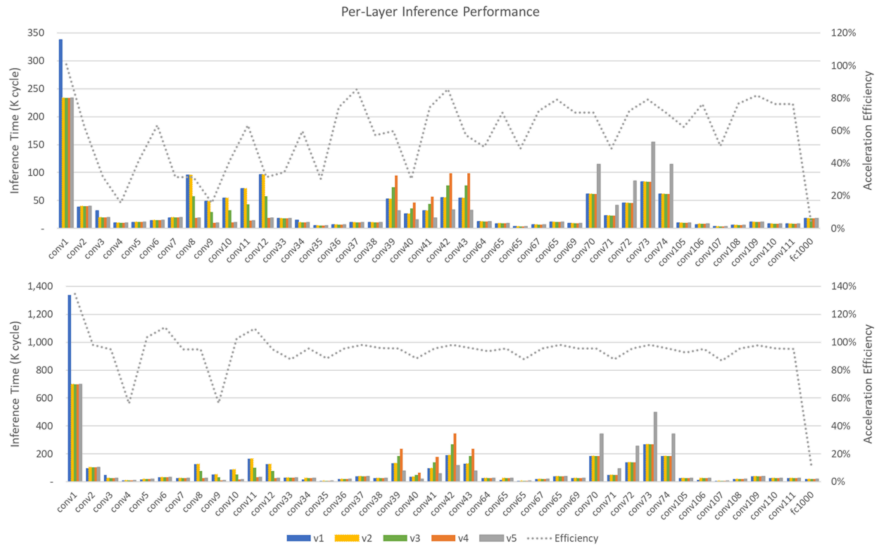


Figure 6: *Per-layer inference time (lower is better) is shown along the left y-axis for variants (v1-v5) of 1.0-SqNxt-23 architecture. Acceleration efficiency (number of MAC operations divided by total cycle counts) is shown by the dotted line and the right y-axis. The top graph shows the results for an* 16×16 *array and the bottom a* 8×8 *PE array. Note the relatively poor efficiency in early layers for the* 16×16 *PE array due to the small number of filter channels.*

Table 3: *Simulated hardware performance results in terms of inference time and energy for the* 8×8 *and* 16×16 *PE array configurations. The time for each configuration is normalized by the number of cycles of the fastest network for each configuration (smaller is better). Note how the variations of the baseline SqueezeNext model are able to achieve better inference and power consumption. For instance, the 1.0-SqNxt-23v5 model is 12% faster and 17% more energy efficient than the baseline model for* 16×16 *configuration. This is achieved by an efficient redistribution of depth at each stage (see Fig. 3). Also note that the 2.0-SqNxt23-v5 has better energy efficiency as compared to MobileNet.*

Model	Params (× 1E+6)	MAC	Top-1	Top-5	Depth	8x8, 32KB Time	8x8, 32KB Energy	16x16, 128KB Time	16x16, 128KB Energy
AlexNet	60.9	725M	57.10	80.30	—	x5.46	1.6E+10	x8.26	1.5E+10
SqueezeNet v1.0	1.2	837M	57.50	80.30	—	x3.42	6.7E+09	x2.59	4.5E+09
SqueezeNet v1.1	1.2	352M	57.10	80.30	—	x1.60	3.3E+09	x1.31	2.4E+09
Tiny Darknet	1.0	495M	58.70	81.70	—	x1.92	3.8E+09	x1.50	2.5E+09
1.0-SqNxt-23	0.72	282M	59.05	82.60	[6,6,8,1]	x1.17	3.2E+09	x1.22	2.5E+09
1.0-SqNxt-23v2	0.74	228M	58.55	82.09	[6,6,8,1]	**x1.00**	2.8E+09	x1.13	2.4E+09
1.0-SqNxt-23v3	0.74	228M	58.18	81.96	[4,8,8,1]	x1.00	2.7E+09	x1.08	2.3E+09
1.0-SqNxt-23v4	0.77	228M	59.09	82.41	[2,10,8,1]	x1.00	2.6E+09	x1.02	2.2E+09
1.0-SqNxt-23v5	0.94	**228M**	**59.24**	82.41	[2,4,14,1]	x1.00	**2.6E+09**	**x1.00**	**2.0E+09**
MobileNet	4.2	574M	67.50(70.9)	86.59(89.9)	—	**x2.94**	9.1E+09	x2.60	5.8E+09
2.0-SqNxt-23	2.4	749M	67.18	88.17	[6,6,8,1]	x3.24	8.1E+09	x2.72	5.9E+09
2.0-SqNxt-23v4	2.56	708M	66.95	87.89	[2,10,8,1]	x3.17	7.5E+09	x2.55	5.4E+09
2.0-SqNxt-23v5	**3.23**	708M	67.44	**88.20**	[2,4,14,1]	x3.17	**7.4E+09**	**x2.55**	**5.4E+09**

Evaluation: Squeezelerator

AI ROBOTICS KR

- **SqueezeNext running on hardware**
  - Acceleration efficacy (# of computation / cycles)
  - **Large filter size applied on a large input feature map (**v1: 7×7 conv, v2: 5×5 conv)
  - Base model (6-6-8-1), v3/v4 (4-8-8-1/2-10-8-1), v5(2-4-14-1)

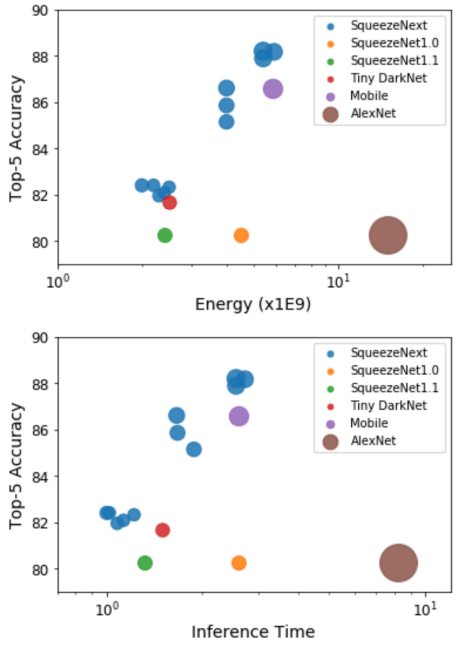


Figure 7: *The spectrum of accuracy versus energy and inference speed for SqueezeNext, SqueezeNet (v1.0 and v1.1), Tiny DarkNet, and MobileNet. SqueezeNext shows superior performance (in both plots higher and to the left is better). The circle areas are proportional to square root of model size for each network.*

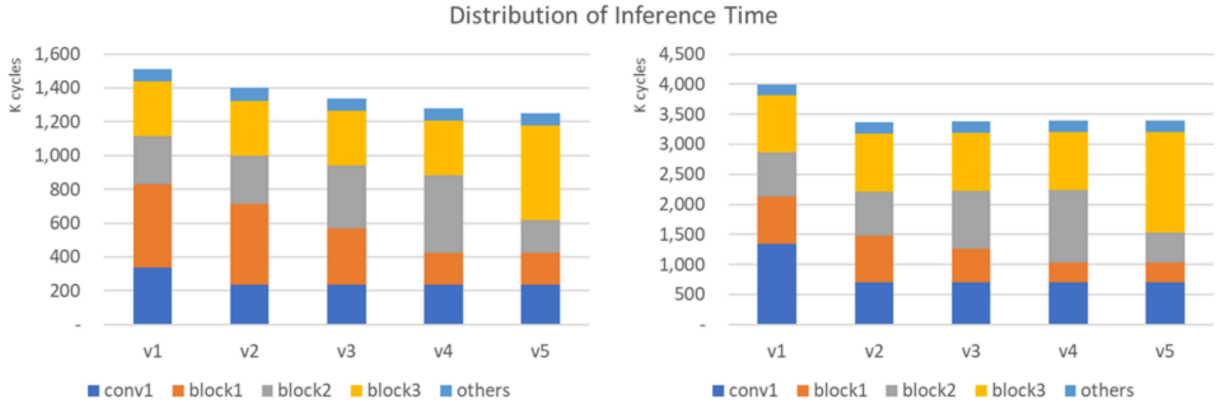


Figure 8: *Distribution of the inference time for the baseline and its optimized variants is shown, along with the contributions corresponding to different modules. Here the modules are broken into blocks that get the same input resolution (height and width). For instance, all layers in block1 have an input feature map of* 55×55*, and block2 gets* 28×28*, etc. The results for the* 16×16 *and* 8×8 *PE configurations are shown in the left and right plots, respectively.*

Discussion

- **Relationship between neural network accelerator and neural network**
  - **Low arithmetic intensity** in depth-separable convolution
  - **Inefficiency** in the computation process