INTEL'S SOLUTION FOR FAST, EFFICIENT INFERENCE

R. Abra, D. Denisenko, R. Allen, T. Vanderhoek
Intel Corporation

ABSTRACT

Convolutional Neural Networks excel at object detection and classification as well as image segmentation; transformers lead the way in sequence analysis, including translation, chatbot and search engine tasks. The value of these algorithms is lost if the inference speed cannot keep up with the application requirements or, just as importantly, power consumption is prohibitively high. We propose a retraining and inference flow that delivers significant reductions in FPGA hardware footprint and/or processing time. Using our solution, we were able to quantize ResNet 50, SSD300 and UNet to 5-bit and 4-bit mantissas with no loss of accuracy, reducing FPGA resources and improving performance.

PROBLEM STATEMENT

A competitive hardware implementation of an AI algorithm must fit to certain parameters. We can look at a realistic use case where:

- ⇒ There are **multiple IPs** on the chip
- ⇒ There is demand for **defined – potentially low – frame latency**
- ⇒ There are **several input streams**
- ⇒ **High accuracy** is essential
- ⇒ **Initial outlay and running costs** are specified

Where this leads to challenges, a number of interdependent parameters reveal tradeoffs that can and often must be made:

Throughput: Throughput corresponds with factors such as network size and precision. On our architecture, the newer networks targeting efficiency (e.g.: EfficientNet, MobileNets) have at least double the framerate capability of the larger networks (e.g.: ResNet50, Inception)

Footprint: Doubling the footprint doubles the throughput but decreases the space for additional functions. However, the footprint can be reduced by lowering inference precision. This may allow choice of a smaller, cheaper chip

Network accuracy: Smaller networks tend to have lower accuracy than larger, more robust architectures. Additionally, after a certain point, precision reduction, advantageous for throughput and on-chip area usage, will compromise accuracy

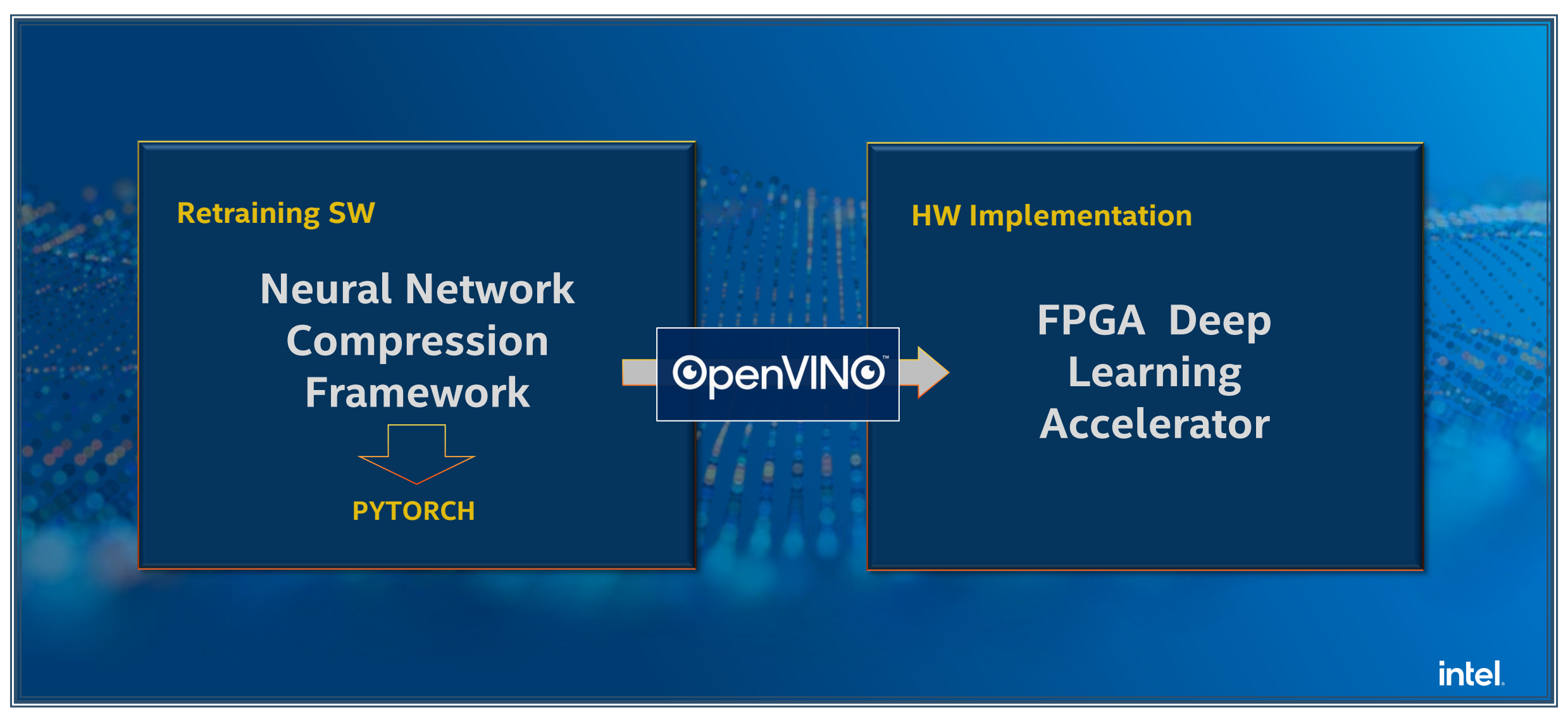
Latency: Smaller networks have lower latency, as do lower precision networks

Our end-to-end quantisation solution provides a way to decrease the architecture footprint and/or processing time as well as latency while maintaining accuracy. In all cases the running costs—and potentially the hardware costs—are reduced.

OUR SOLUTION

Our solution splits neatly into three parts:

1. **Neural Network Compression Framework.** We add block floating point quantisation for use with the the FPGA AI Suite. Optional retraining over a few epochs (<13) allows dropped accuracy from lowering the precision to be regained (see table to the right)
2. **OpenVINO** provides the following:
   - Model Optimiser converts and optimises the network: batch norm layers are folded into convolutional layers for faster, platform-agnostic inference
   - Plugins for the specified accelerators
3. **FPGA AI Suite** including the Deep Learning Accelerator. A parameterisable soft IP core with overlay architecture that is customisable for a specific network or networks. Developed for Intel's Arria 10 and Agilex FPGAs

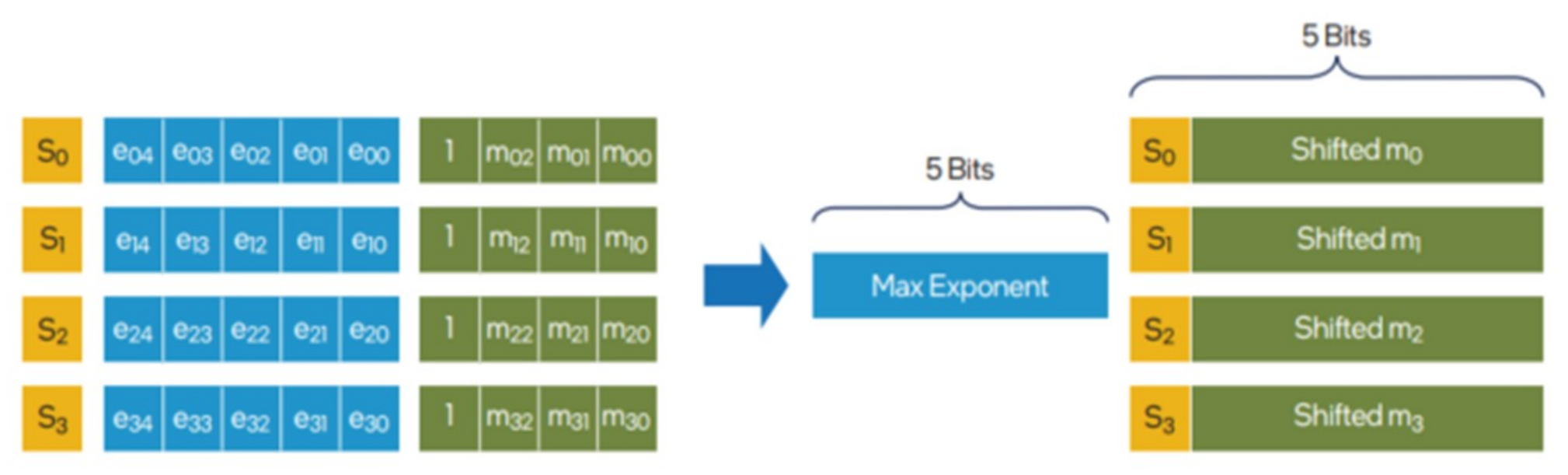

Retraining SW
Neural Network Compression Framework
PYTORCH
OpenVINO
HW Implementation
FPGA Deep Learning Accelerator
intel

Our three-part complete solution for efficient inference: *NNCF for low precision retraining, OpenVINO for format conversion and DLA for hardware-specific inference*

The DLA is innately tuned for block floating point. Here, floating-point numbers are organized into groups then arranged to share a common exponent. The block size is parameterised, denoting the number of mantissas sharing the exponent. Tuning block size can be used to trade off throughput for footprint savings.

WHY BLOCK FLOATING POINT?

The image below shows the conversion of floating point to block floating point. The "1" in the left-most mantissa positions is the implicit 1 in floating-point format made explicit prior to conversion. Sign + mantissa bits are now in sign + magnitude integer format.

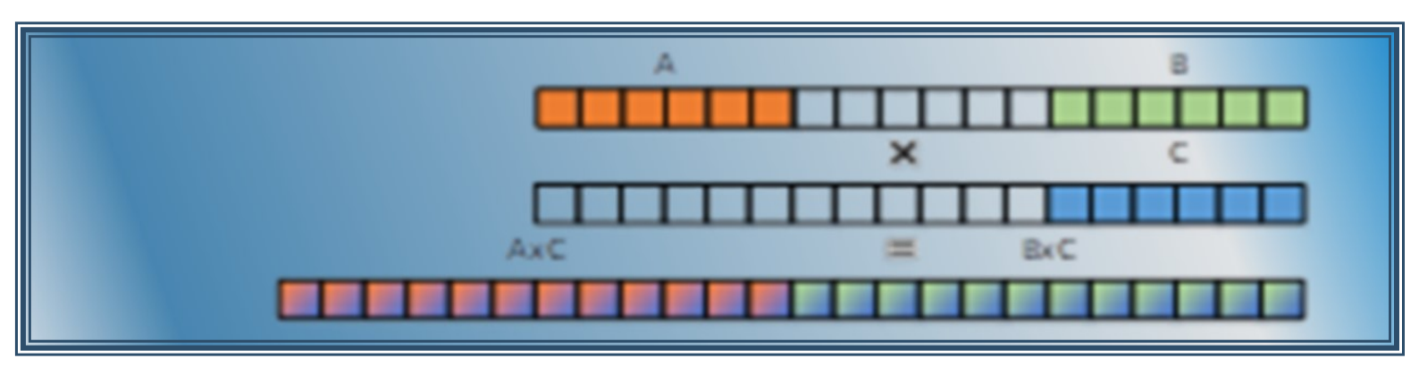

S0 e04 e03 e02 e01 e00 1 m02 m01 m00
S1 e14 e13 e12 e11 e10 1 m12 m11 m10
S2 e24 e23 e22 e21 e20 1 m22 m21 m20
S3 e34 e33 e32 e31 e30 1 m32 m31 m30
5 Bits
Max Exponent
5 Bits
S0 Shifted m0
S1 Shifted m1
S2 Shifted m2
S3 Shifted m3

Conversion of floating point to block floating point: *blocking of four FP9 floating point values to int5bfp*

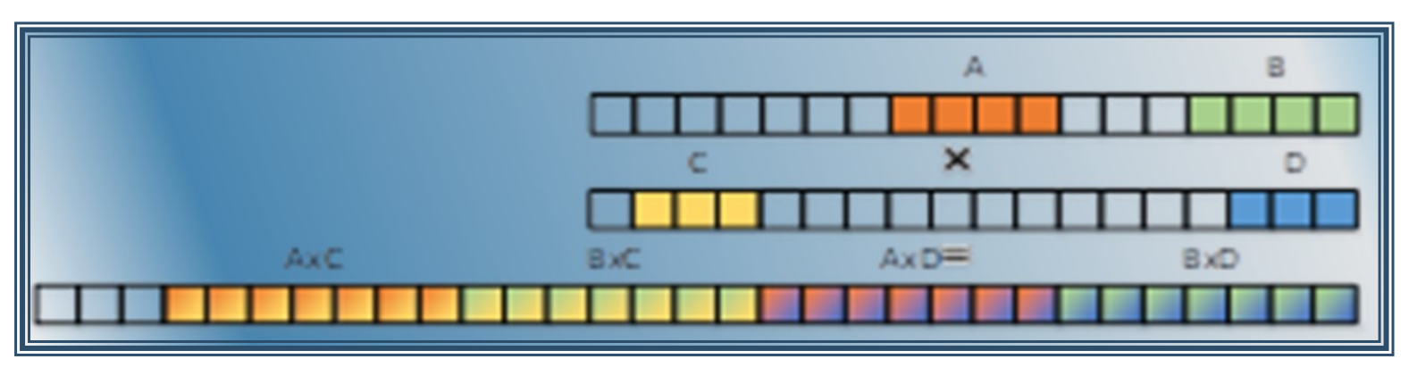
NB our nomenclature uses the format intXbfp where X represents the mantissa plus sign bit. Int7bfp, for example, has 6 mantissa bits plus the sign bit. All values have five exponent bits.

Certain quantisation precisions work very well on specific devices. For example the Arria 10 DSP blocks contain 18-bit multipliers with 24-bit accumulators. To make full use of these, we can provide:

- A single 18-bit x 18 –bit multiply
- Two 6-bit x 6-bit (int7bfp) multiplies
- Four 4-bit x 3-bit (int5/4bfp) multiplies



Two 6-bit x 6-bit multiplies



Four 4-bit x 3-bit multiplies

This optimisation is highly advantageous in the case of convolutional neural networks. CNNs are compute-intensive but at specific precisions, the mantissas in the simple matrix-matrix multiplications can be packed efficiently into the DSP blocks as above.

Equally importantly, block floating point has a high dynamic range which models weights and activations well at low precision.

CONCLUSIONS

Many AI applications, particularly in the embedded market, have stringent requirements for power and performance. These are often complicated by extra functions needed on the chip. A great benefit of FPGAs is that functions such as I/O, clipping, scaling and warping can all be included on the same chip as the network.

When used for streaming, this configuration reduces the need for costly frame buffering and off-chip memory transfers.

Block floating point itself has two major advantages:

- ⇒ It transfers very well to Intel FPGAs, particularly at certain sizes. Efficient DSP block packing allows the footprint reductions seen, taking the pressure off the ALMs and the need for storage in RAM
- ⇒ It has a high dynamic range, making it adept at retaining and regaining accuracy even t very low precision. A default block size of 32 was sufficient to allow larger networks to quantise down to int7bfp without retraining

On leaner, more compact topologies such as MobileNets and EfficientNet, the resultant drop at low precision was overcome by retraining for a few epochs.

The software itself has several benefits. In addition to providing a (re)training facility, it allows the testing of quantising a network for accuracy before compiling.

ACCURACY RESULTS

			Int5/4bfp			*Int7bfp*			*Int8bfp*	
Network	**FP32 accuracy reference (%)**		**Accuracy (%) without re-training**	**Accuracy (%) with retraining**		**Accuracy (%) without re-training**	**Accuracy (%) with retraining**		**Accuracy (%) without re-training**	**Accuracy (%) with retraining**
Classification (ImageNet)										
ResNet-18	69.76		55.69	69.13		69.67	n/a*		69.60	n/a
ResNet-34	73.31		65.09	72.81		72.94	n/a		73.09	n/a
ResNet-50	76.13		60.32	75.60		75.75	n/a		75.95	n/a
Inception V3	77.32		32.70	78.34		77.11	n/a		77.31	n/a
EfficientNet_b0	75.86		0.34	71.96		64.37	75.45		70.48	75.47
MobileNet V2	71.81		6.00	67.90		67.28	71.65		71.12	n/a
SqueezeNet V1.1	58.18		33.09	54.90		57.73	58.15		58.10	n/a
Object Detection (VOC 2007 & 2012)										
SSD300	78.12		73.64	77.92		78.09	n/a		78.08	n/a
SSD512	80.26		74.72	80.00		80.19	n/a		80.08	n/a
Object Detection (COCO 2017)										
TinyYOLO v3	35.7		26.90	31.40		35.50	n/a		35.60	n/a
Semantic Segmentation (CamVid)										
UNet	71.95		63.95	72.36		71.66	n/a		71.89	n/a
ICNet	67.89		59.66	67.09		67.88	n/a		67.87	n/a

Indicative accuracies for networks both with and without retraining *at int5/4bfp (int5bfp activations and int4bfp weights), int7bfp and int8bfp. n/a shows where retraining is not required*

Several conclusions can be drawn from the table above:

1. Most networks quantise well to int8bfp and even int7bfp without retraining (green cells)
2. This accuracy drop is much more pronounced at very low precision (int5/4bfp)
3. The accuracy drop was exaggerated the more compact the network (EfficientNet, MobileNet, SqueezeNet)
4. In most cases, any loss was recoverable on retraining with fewer than 12 epochs ("Accuracy with retraining" columns)

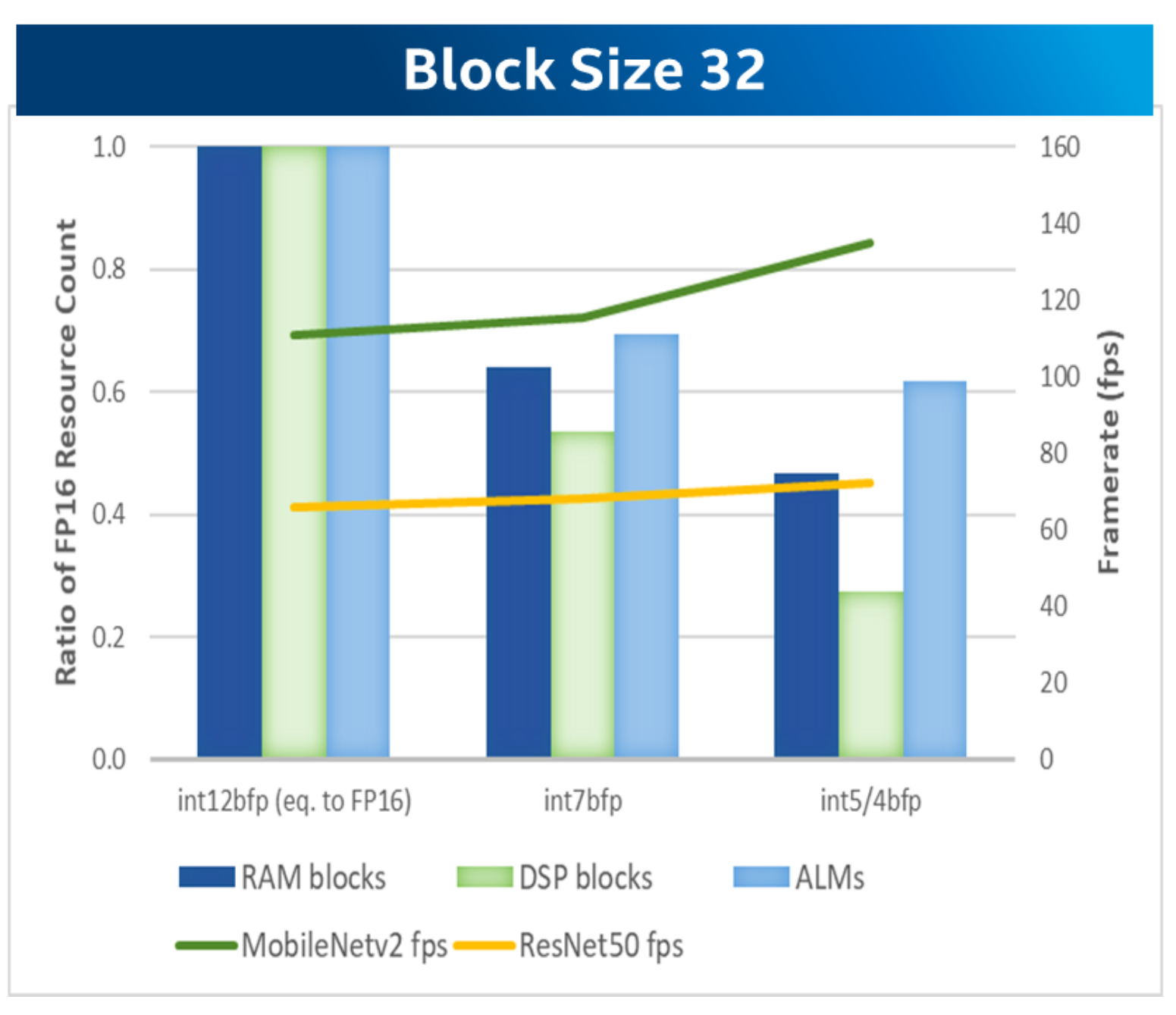
It was successfully shown that retraining in block floating point allows augmentation of accuracy to FP32 levels in the vast majority of network and precision choices. This gives additional flexibility for real-world implementation.

NOTES:

Accuracy was regained at very low precision only when activations were represented with at least as many bits as the weights.

For all cases above, block size is a nominal 32. Inference on hardware confirmed the classification results at both int7bfp and int8bfp. At the time of investigation, hardware was not available for validation of lower precisions. Validated was using an Intel® Programmable Accelerator Card with Intel® Arria® 10 FPGA in a desktop HP® Z640 machine.

HARDWARE TESTING



Resource count ratios and framerate *for ResNet 50 and MobileNet v2 inference at block size 32 on Arria® 10*

Measurable benefits come in the form of reductions in hardware resource usage that is derived from the lower precisions.

The overlay structure of the DLA supports both MobileNet v2 and ResNet 50 on the same architecture, so we can show both on the same graph.

At block size 32, we show that precision reduction, as per the efficient packing methods described, roughly halves the hardware resources each time. In select cases the throughput goes up.

The graphs use int12bfp block size 32 as the baseline reference, which requires the following resource counts:

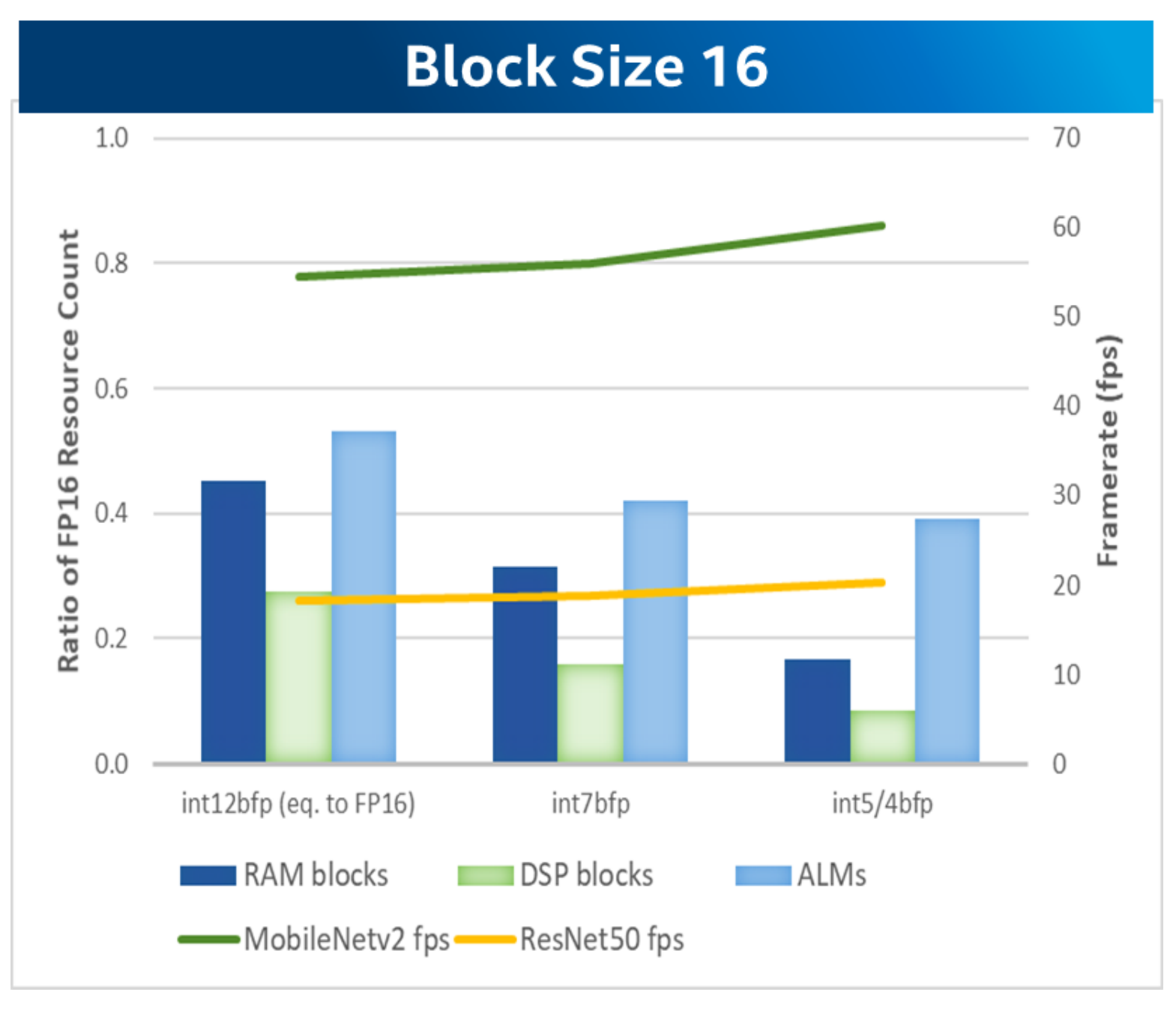
816 M20k RAM blocks

551 DSP blocks

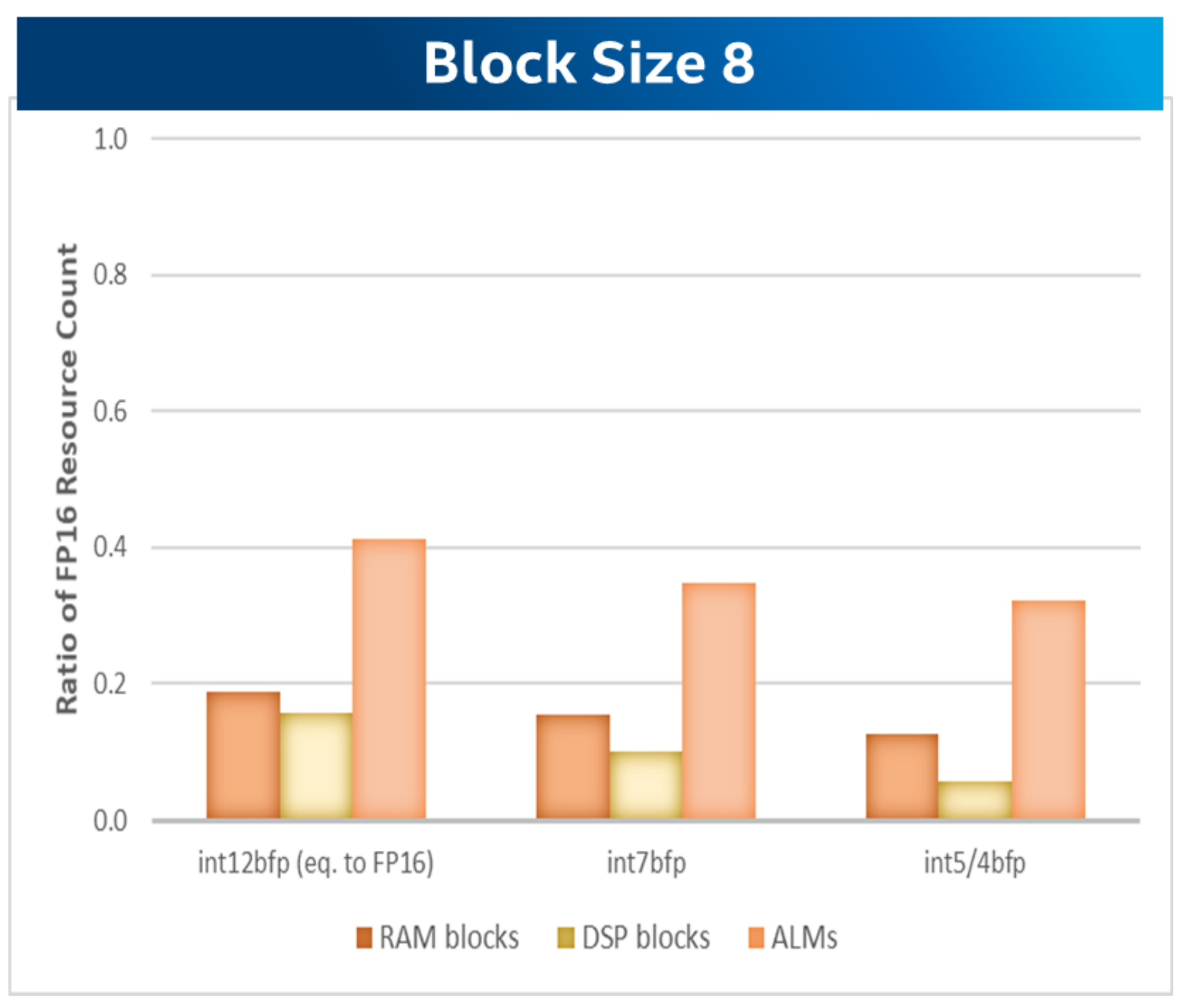
39k ALMs

Re-optimisation of the buffers and bus sizes allows an additional trade-off, this time between resource usage and throughput. Reducing the block size to 16 gives the same decrease in resources seen above, where resource counts are a fraction of those we started with. DSP block usage is a quarter of the block size 32 usage due to the two-dimensional reduction in calculations from both weights and activations.

Unsurprisingly, the throughput is also reduced. MobileNet's is halved, while ResNet's throughput is less than one third its starting point. The latter contains many very small convolutions which only add to the cumbersomeness.



Resource count ratios and framerate *for ResNet 50 and MobileNet v2 inference at block size 16 on Arria® 10*



Resource count ratios and framerate *for MobileNet v2 inference at block size 8 on Arria® 10*

Finally, we give a realistic example of an application that requires MobileNet v2 running at a framerate of 30fps. In this case, a block size of 8 is sufficient, which reduces the footprint significantly. The final figures for int5/4bfp are:

103 M20k RAM blocks

31 DSP blocks

13k ALMs

Interestingly, the Intel® Agilex® DSP block provides the ability to pack four 9-bit x 9-bit multiplies. When Agilex DSP blocks are chained together, this is equivalent to absorbing the dot product adder tree into the DSP and out of ALM logic. The result is a significant reduction in ALM logic when the chosen precision allows use of this mode. Even if floorplan and placement considerations make it undesirable to create long DSP chains, this still allows absorption of significant portions of the adder tree. This, combined with the Fmax increase, will affect the precision and area trade-offs. Future work will investigate these trade-offs on the Agilex® architecture.