Low-Precision Floating-Point for Efficient On-Board Deep Neural Network Processing

Cédric Gernigon, Silviu-Ioan Filip, Olivier Sentieys, Clément Coggiola, Mickaël Bruno

Low-Precision Floating-Point for Efficient On-Board Deep Neural Network Processing

Cédric Gernigon
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes, France
Email: cedric.gernigon@inria.fr

Silviu-Ioan Filip
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes, France
Email: silviu.filip@inria.fr

Olivier Sentieys
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes, France
Email: olivier.sentieys@inria.fr

Clément Coggiola
Spacecraft techniques, on-board data handling
CNES, Toulouse, France
Email: clement.coggiola@cnes.fr

Mickaël Bruno
Spacecraft techniques, on-board data handling
CNES, Toulouse, France
Email: mickael.bruno@cnes.fr

***Abstract*—One of the major bottlenecks in high-resolution Earth Observation (EO) space systems is the downlink between the satellite and the ground. Due to hardware limitations, on-board power limitations or ground-station operation costs, there is a strong need to reduce the amount of data transmitted. Various processing methods can be used to compress the data. One of them is the use of on-board deep learning to extract relevant information in the data. However, most ground-based deep neural network parameters and computations are performed using single-precision floating-point arithmetic, which is not adapted to the context of on-board processing. We propose to rely on quantized neural networks and study how to combine low precision (*mini*) floating-point arithmetic with a Quantization-Aware Training methodology. We evaluate our approach with a semantic segmentation task for ship detection using satellite images from the Airbus Ship dataset. Our results show that 6-bit floating-point quantization for both weights and activations can compete with single-precision without significant accuracy degradation. Using a Thin U-Net 32 model, only a 0.3% accuracy degradation is observed with 6-bit minifloat quantization (a 6-bit equivalent integer-based approach leads to a 0.5% degradation). An initial hardware study also confirms the potential impact of such low-precision floating-point designs, but further investigation at the scale of a full inference accelerator is needed before concluding whether they are relevant in a practical on-board scenario.**

***Index Terms*—Deep Neural Networks (DNN), Reduced Precision, Quantization-Aware Training (QAT), Floating-Point, Ship Detection, Semantic Segmentation**

I. Introduction

Earth Observation (EO) provides an effective way of exploring the physical, chemical, and biological information related to the Earth. This information collected by EO satellites is widely used in various research fields, especially in relation to the environment, where the measurements made by EO satellites are indispensable. Moreover, these new space applications related to Earth observation produce a huge volume of data extracted from various image and radar sensors. Transmitting all this data is possible through communication between satellites and ground stations. However, EO systems are limited by these downlink communications, due to hardware limitations, on-board power constraints or ground-station operation cost, for example. Thus, there is a need to reduce the amount of data to be transmitted through the downlink. While data compression is widely used for size reduction, the idea of transmitting only relevant data through on-board processing has only recently started gaining interest.

Artificial Intelligence (AI), and in particular Deep Learning (DL), is starting to be successfully applied in space applications. However, the inference computation of many models is still mainly performed on ground platforms due to their memory footprint and computational intensity [12]. To mitigate this computational burden and the space-to-ground communication bottleneck, recent research has focused on neural network compression [6]. Moreover, various techniques such as pruning [14], weight sharing [10], distillation [9] and quantization [17], can be used to reduce the computational intensity to make it compatible with on-board processing.

Quantization deals with *how* (what number format(s) and bit widths to use) model parameters (such as weights and bias terms) and activation signals (inputs and outputs to layers in a model) are computed and stored at inference time. Furthermore, reducing precision and adapting the number representation make quantization a particularly effective and practical technique.

In this paper, we investigate how to efficiently use quantization to accelerate Deep Neural Network (DNN) inference for space applications, with a strong focus on semantic segmentation tasks. Towards this goal, we present our ongoing work on the efficient use of low-precision floating-point quantization (so-called *minifloat*s). Our method consists in adapting a quantized DNN training approach that has so far been mostly used for integer/fixed-point-based quantization.

II. Background and Related Work

Deep neural network parameters and computations have been traditionally stored and computed using 32-bit single-

This work is partly supported by the French Space Agency (CNES), Institut National de Recherche en Informatique et en Automatique (INRIA), and the COMINLabs (10-LABX-0007) LeanAI project.

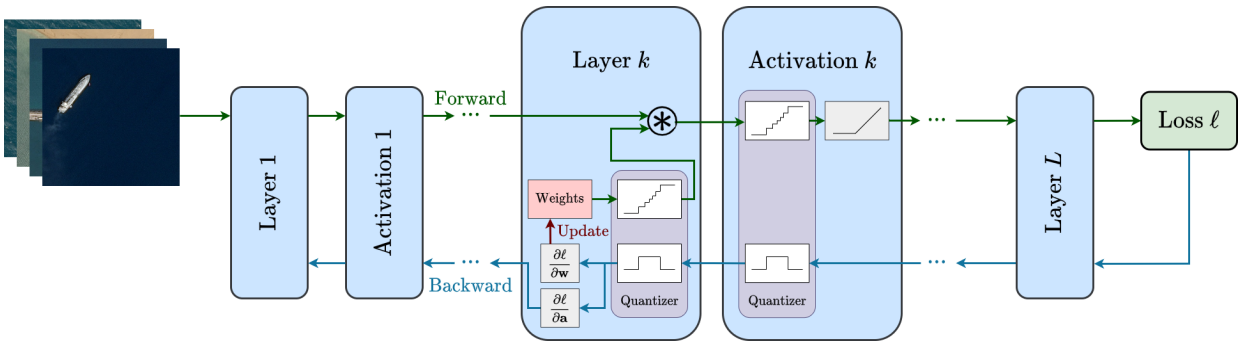


Fig. 1: An overview of the QAT scheme we use in the training of minifloat quantized DNNs. The distinguishing feature of a QAT flow compared to a standard training procedure is the presence of quantizer blocks in each layer and activation function that use the STE mechanism to differentiate with respect to the quantized signals.

precision floating-point (FP) arithmetic, making it impossible to deploy state-of-the-art models on low-power and resource-constrained devices without further tuning.

A. DNN Quantization

Much work in recent years has focused on reducing the bit-width of data and arithmetic in DNN models without impacting task accuracy. Quantization to 8-bit [18] and sub 8-bit [34] integer formats has been shown to match single-precision baselines on convolutional networks for several computer vision tasks. In certain cases, it is even possible to go down to extremely low precisions, such as binary [8] and ternary [23] DNNs.

On the FP arithmetic side, 16-bit formats such as half precision FP16 (5-bit exponent) and bfloat16 (8-bit exponent) have found success in both inference acceleration and mixed-precision DNN training [19]. More recently, 8-bit floating-point formats have also been explored in inference/training scenarios [21], [25], [32] and are starting to get hardware support (*e.g.* the NVIDIA Hopper architecture is an example). Sub 8-bit FP custom formats are also being studied in the literature in the context of model compression and inference acceleration [28].

To apply such quantization schemes on a DNN model, it is necessary to adjust the parameters according to the target format. There are two main approaches to do this in practice. The first is Post-Training Quantization (PTQ), in which quantization is applied after training. While fast, it can lead to non-negligible loss in accuracy for extremely low bit width formats [2]. The other approach is to iteratively quantize the model during training. This process is known as Quantization-Aware Training (QAT). While much slower, it generally leads to better quantization results. For an in-depth overview of quantization in the context of DNNs the reader can consult [13].

B. Quantization-Aware Training

The emergence of QAT methods can be traced back to the pioneering work of [7], [8] on binary quantization of DNNs. The overall approach consists of using the quantized version of the network during forward and backward computations done throughout training, while performing updates on a full precision copy of the parameters (*e.g.* weights and biases). A critical ingredient of these methods is the use of a so-called Straight-Through Estimator (STE) [3] that allows the back-propagation of gradients with respect to quantized binary variables (weights and activation signals). Subsequent work [34] extends these ideas to larger bit widths and tackles quantization of gradient signals as well. Other methods propose learning parameters that bound the range of activation signals inside the network [5] and new gradient estimates that allow learning of appropriate scaling factors for integer-based quantization of both weights and activations [11].

III. Methodology

A. Floating-point Encoding

The floating-point formats we use, denoted as EeMm, are based on the IEEE-754 standard [1]. A floating-point value X with m bits of mantissa and e bits of exponent is represented (in binary notation) as

$$X = (-1)^s \times 1.\underbrace{x_1 \dots x_m}_{M_X} \times 2^{E_X - E_B},$$

where s is the sign bit, M_X is the m-bit fractional mantissa ($M_X \in [0, 1)$) and $E_X \in [0, 2^e - 1]$ is the integer exponent. E_B is the exponent bias term, which in the case of IEEE-like encodings is $E_B = 2^{e-1} - 1$. With respect to a fully IEEE-compliant format, we do not support $\pm\infty$, `NaN` encodings, and subnormal values. This leads to more representable values and simpler hardware (see [24]). Values that overflow are saturated to the maximal representable number, whereas zeros are represented by $E_X = 0$ and $M_X = 0$. The values that would have been treated as subnormals ($E_X = 0, M_X > 0$ and $X = (-1)^s \times 0.M_x \times 2^{-E_B}$) are instead viewed as one extra binade of normal values.

The actual exponent $E_X - E_B \in [-2^{e-1} + 1, 2^{e-1}]$, as described in the IEEE-754 standard, covers an almost symmetric range of positive and negative values. If the dynamic range of the data to quantize is known, E_B can be adapted

to better match it. The PTQ `AdaptivFloat` [28] method does this by examining the maximum magnitude of the weight tensors in each layer of the network. In a QAT setting, E_B can be better chosen by learning it using a STE [3] approach. Just as for the scaling factor of an integer-based format, a similar method [21] can be used to learn a real bias exponent in the floating-point case. In order to achieve a more hardware-friendly minifloat format, we propose to further quantize this learned real exponent bias to an integer. We explore this in Sec. III-B.

B. Quantization Scheme

Our work can be seen as an extension of [21]. While initially considered for FP8 quantization with subnormals and a real exponent bias, we investigate its use in a sub 8-bit FP context without subnormals and an integer exponent bias. Subnormal support adds some hardware overhead, but for minifloat formats with small mantissa (2 or 3 bits), using the subnormal range as normal values can still lead to good results with a much smaller overhead (see [29]).

We quantize weights and activations using an STE-based approach as follows:

$$\textbf{Forward: } \mathbf{X}_q = \texttt{quantize}\left(\mathbf{X}, E_0\right)$$
$$\textbf{Backward: } \frac{\partial \ell}{\partial \mathbf{X}} = \frac{\partial \ell}{\partial \mathbf{X}_q}\frac{\partial \mathbf{X}_q}{\partial \mathbf{X}}$$

where $\mathbf{X}$ is an unquantized weight or activation signal, ℓ is the loss function, and $\mathbf{X}_q$ is the EeMm quantized version of $\mathbf{X}$ computed using Algorithm 1. The straight-through moniker comes from the fact that we take $\partial \mathbf{X}_q / \partial \mathbf{X} = 1$.

We propose to learn the exponent biases of each layer's weights and activations during training. The learned real values are rounded up (line 1 of Algorithm 1) to scale the network signals (weights & activations) by powers of two. The real values are stored for future use in the update phase with a SGD-type procedure (going from an iteration t to $t+1$) as summarized in Algorithm 2 (lines 18 and 19). The g quantities in lines 8–13 represent the gradients of the loss ℓ with respect to the activation, weight and exponents, respectively. Their computation (the `backward` function calls) is handled through the PyTorch autograd engine. A schematic view of the entire QAT iteration is also given in Figure 1.

IV. Experiments

We apply our approach on a image classification problem on the CIFAR-10 dataset [20] and on a satellite image segmentation task using a lightweight U-Net model (adapted from [30]).

A. CIFAR-10 Dataset and Implementation Details

The CIFAR-10 dataset consists of 32×32 pixels RGB labeled images divided into 10 categories. The dataset is composed of $50,000$ training images and $10,000$ test images. We trained a ResNet-20 model [16] from scratch for 300 epochs using a cross entropy loss function. We apply the data augmentation operations proposed in [22] over the training set and we use an SGD optimizer with 128 batch size, weight decay set to 0.0001, and momentum to 0.9. Weights are initialized using the Kaiming method [15] and the learning rate is scheduled using cosine annealing with starting rate set to 0.1.

Algorithm 1 `quantize`: minifloat quantization algorithm

Require: real-valued tensor $\mathbf{X}$, floating-point format EeMm, real learned exponent bias E_0.
Ensure: quantized tensor $\mathbf{X}_q$ in the EeMm format.

1: $E_B = \lceil E_0 \rceil$
2: $x_{\min} = 2^{-E_B} \cdot (1 + 2^{-m})$
3: $x_{\max} = 2^{2^e - 1 - E_B} \cdot (2 - 2^{-m})$
4: $\mathbf{X}_\text{c} = \texttt{clamp}\left(\mathbf{X}, -x_{\max}, x_{\max}\right)$
5: $\mathbf{X}_\text{scale} = 2^{\lfloor \log_2(\mathbf{X}_\text{c}) \rfloor - m}$
6: $\mathbf{X}_\text{q} = \left\lfloor \frac{\mathbf{X}_\text{c}}{\mathbf{X}_\text{scale}} \right\rceil \cdot \mathbf{X}_\text{scale}$
7: $\mathbf{X}_\text{q} = \mathbf{X}_\text{q} \cdot \mathbb{I}_{|\mathbf{X}_\text{q}| \geq x_{\min}}$
8: **return** $\mathbf{X}_q$

Algorithm 2 FP QAT algorithm for training a L-layer model

Require: a minibatch of inputs $\mathbf{Y}_0^t$ and targets $\mathbf{T}^t$, weights $\mathbf{W}^t \in \mathbb{R}$, activation exponent biases $E_{0,a}^t$, weight exponent biases $E_{0,w}^t$, learning rate $\eta > 0$, loss function ℓ.
Ensure: updated parameters $\mathbf{W}^{t+1}$, $E_{0,w}^{t+1}$ and $E_{0,a}^{t+1}$

1: **1. Forward propagation:**
2: **for** $k = 1$ **to** L **do**
3: $\quad \mathbf{W}_{q,k}^t \leftarrow \texttt{quantize}\left(\mathbf{W}_k^t, E_{0,w,k}^t\right)$
4: $\quad \tilde{\mathbf{Y}}_k^t \leftarrow \texttt{forward}\left(\mathbf{Y}_{q,k-1}^t, \mathbf{W}_{q,k}^t\right)$
5: $\quad \mathbf{Y}_k^t \leftarrow \texttt{quantize}\left(\tilde{\mathbf{Y}}_k^t, E_{0,a}^t\right)$
6: **end for**
7: **2. Backward propagation:**
8: $g_{\mathbf{Y}_L^t} = \frac{\partial \ell\left(\mathbf{Y}_L^t, \mathbf{T}^t\right)}{\partial \mathbf{Y}_L^t}$
9: **for** $k = L$ **down to** 1 **do**
10: $\quad g_{\mathbf{Y}_{k-1}^t} \leftarrow \texttt{backward\_activ}\left(g_{\mathbf{Y}_{q,k}^t}, \mathbf{W}_{q,k}^t\right)$
11: $\quad g_{\mathbf{W}_k^t} \leftarrow \texttt{backward\_weight}\left(g_{\mathbf{Y}_{q,k}^t}, \mathbf{Y}_{q,k}^t\right)$
12: $\quad g_{E_{0,a,k}^t} \leftarrow \texttt{backward\_exp\_bias\_a}\left(g_{\mathbf{Y}_{q,k}^t}, \mathbf{Y}_{q,k}^t\right)$
13: $\quad g_{E_{0,w,k}^t} \leftarrow \texttt{backward\_exp\_bias\_w}\left(g_{\mathbf{Y}_{q,k}^t}, \mathbf{Y}_{q,k}^t\right)$
14: **end for**
15: **3. Parameter update:**
16: **for** $k = 1$ **to** L **do**
17: $\quad \mathbf{W}_k^{t+1} \leftarrow \texttt{update}\left(\mathbf{W}_k^t, g_{\mathbf{W}_k^t}, \eta\right)$
18: $\quad E_{0,w,k}^{t+1} \leftarrow \texttt{update}\left(E_{0,w,k}^t, g_{E_{0,w,k}^t}, \eta\right)$
19: $\quad E_{0,a,k}^{t+1} \leftarrow \texttt{update}\left(E_{0,a,k}^t, g_{E_{0,a,k}^t}, \eta\right)$
20: **end for**

B. Airbus Ship Dataset and Implementation Details

The Airbus Ship Dataset[1] consists of 768×768 pixels RGB satellite images of ships with sea and harbor in the back-

[1] Kaggle Airbus Ship Detection Challenge (July 2018): https://www.kaggle.com/c/airbus-ship-detection

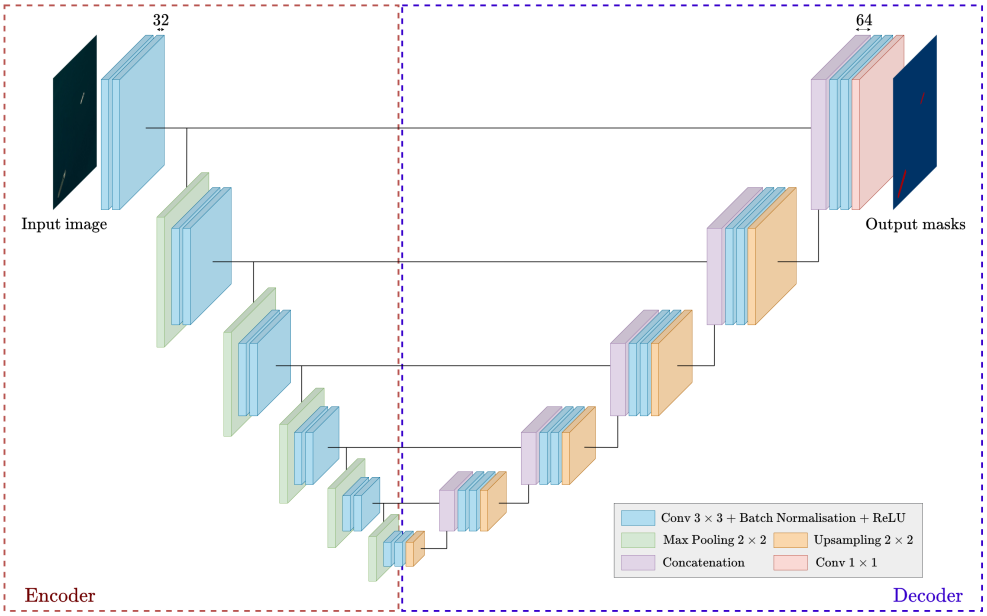


Fig. 2: Thin U-Net 32 architecture. It consists of a 5-stage encoder, followed by a 5-stage decoder, with skip connections between each corresponding stage pairs in the encoder-decoder blocks.

ground. The training data is composed of $192,555$ labelled images, including $42,555$ images with ships and $150,000$ background images. To balance the data, we removed $130,000$ background images and split the remaining $62,555$ images into 80% for training and 20% for testing.

We use a Thin U-Net [30] model (see Fig. 2), a smaller version of the larger and more widely known U-Net [27] architecture. More specifically, we consider the Thin U-Net 32 model, where 32 refers to the number of channels for each convolution. Its memory size is smaller by a factor of 290 compared to a standard U-Net architecture (going down from 288.08Mb to 0.99Mb), with little impact on accuracy. Similar to the standard U-Net, this model consists of two distinct blocks, an encoder and a decoder, interconnected by skip connections. The encoder is composed of 5 blocks containing two groups of 3×3 convolution + batch normalisation + ReLU with 2×2 maxpooling to down sample the feature maps. Followed by the decoder, it also has 5 blocks containing a 2×2 bilinear up-sampling layer and two groups of 3×3 convolution + batch normalisation + ReLU. Prediction masks are generated by a final 1×1 convolution without padding.

We apply random horizontal and vertical flip, random crop to resize images to 256×256 pixels, random brightness and random contrast as data augmentations on the training set. We use ADAM as the optimizer with a batch sizes of 32 and an aggregated global loss consisting of a Jaccard loss combined with binary cross entropy with logits, which according to [31] gives the best results in practice. The learning rate is initially set to 0.001 and we use a multi step scheduler in order to reduce the learning rate by 0.5 every 200 epochs. We train the network for 600 epochs, using the Kaiming weight initialization method [15]. For computing the quantized model,

TABLE I: Comparison of prediction accuracy for CIFAR-10 with different arithmetic formats and bit-widths for weights (W) and activations (A).

Arithmetic Format	Top-1	W bit-width	A bit-width
Single precision	92.5	M23E8	M23E8
Fixed-point	90.2	3	3
Integer	91.6	3	3
Minifloat	91.3	M1E1	M2E2
	91.3	M1E2	M1E2
	90.8	M1E1	M1E2

we used the QAT approach from Algorithm 2 to fine tune the pretrained single-precision model for 50 epochs.

Just like with an integer scaling factor [4], a good initialization of the exponent bias parameter is key to convergence with good accuracy. We have found that in practice good initial estimates can be determined by first training the network for a small number of iterations without optimizing the exponent bias (we did this for 200 iterations in our tests) and then picking it based on the values with maximum magnitude in the weight and activation tensors seen during this process, leading to

$$E_0 = 2^{e-1} - \left\lceil \log_2 \left(\frac{\max |\mathbf{X}|}{2 - 2^{-m}} \right) \right\rceil .$$

C. Results

We use PyTorch 1.13 on a cluster of eight NVIDIA V100 GPUs to perform our experiments. Unlike the majority of integer-based quantization methods described in the literature which quantize the first and last layers to 8 bits (which is

TABLE II: Comparison of prediction accuracy for the Thin U-Net 32 model on the Airbus Ship dataset with different arithmetic formats and bit-widths.

Format	mean IoU	W bit-width	A bit-width	scaling factor	zero encoding
Single-precision	71.0	M23E8	M23E8	/	$M_X = 0$ and $E_X = 0$
Fixed-point	44.5	6	6	$2^{\lceil \log_2(\max \lvert X \rvert) \rceil}$	Zero point $= 0$
Integer	70.5	6	5	learn	Zero point $= 0$
	68.3	5	4	learn	Zero point $= 0$
Minifloat	63.4	E3M2	E3M2	$2^{2^{e-1}}$	$E_X = 0$
	64.8	E3M2	E3M2	$2^{2^{e-1}}$	$M_X = 0$ and $E_X = 0$
	70.1	E3M3	E3M3	learn	$E_X = 0$
	70.0	E3M2	E3M2	learn	$E_X = 0$
	71.4	E4M2	E4M2	learn	$M_X = 0$ and $E_X = 0$
	70.9	E3M3	E3M3	learn	$M_X = 0$ and $E_X = 0$
	70.7	**E3M2**	**E3M2**	**learn**	$\mathbf{M_X = 0}$ **and** $\mathbf{E_X = 0}$
	68.1	E2M2	E2M2	learn	$M_X = 0$ and $E_X = 0$

usually a larger word length than that for the other layers), we use the same small length format for all layers. This can lead to slightly better compression ratios, without affecting accuracy.

We have compared our custom floating-point formats with fixed-point and integer arithmetic alternatives, two quantization formats commonly used to accelerate inference. Results on the CIFAR-10 dataset are given in Table I. While fixed-point quantization seems to degrade model accuracy significantly, low-precision floating-point variants are competitive with integer-based alternatives for very low quantization levels.

The results of applying minifloat quantization to ship detection are summarized in Table II. On this more complex task, 6-bit floating-point formats are necessary to match single precision accuracy, which is evaluated using an *Intersection over Union* (IoU) metric.

We note that a visual analysis of prediction outputs for the minifloat quantized model shows results that are relatively close to the single-precision baseline. When single ships are present in the image, our quantized model usually does a good job of detecting them (Fig. 3). Detecting small ships, as well as side by side ships and inshore ships is more challenging than detecting single large ones, leading to poorer predictions even with single-precision models (Fig. 4 and Fig. 5). Inshore [26] and small ship [33] detection are challenging topics, both subject to active research.

D. Hardware Implementation Aspects

The results we have shown so far suggest that minifloat quantization is potentially a good choice for low-precision inference acceleration. However, floating-point addition is in general more resource-intensive than its integer/fixed-point counterpart.

This is somewhat counterbalanced by the multiplier, which in case of minifloats can be implemented efficiently using just look-up tables (LUTs), as opposed to an 8-bit integer variant that would require DSP blocks, which are much lower in number than LUTs on modern FPGAs (*e.g.* in a AMD-Xilinx UltraScale+ VUP13 FPGA, for every DSP block there are 140 6-input LUTs [24]). DSPs can then be configured to implement adder trees for the accumulation part of multiply-accumulate (MAC) units, improving overall logic density. For instance, the results presented by AMD-Xilinx in [24] claim 60% higher performance and 12.5% memory traffic reduction (leading also to lower power usage) when using a minifloat E3M3 format as opposed to INT8 in a ResNet-50 accelerator. To achieve this performance, they implemented as basic building block a hybrid MAC operator that combines the best of both worlds, a LUT-based minifloat multiplier and a simpler fixed-point adder.

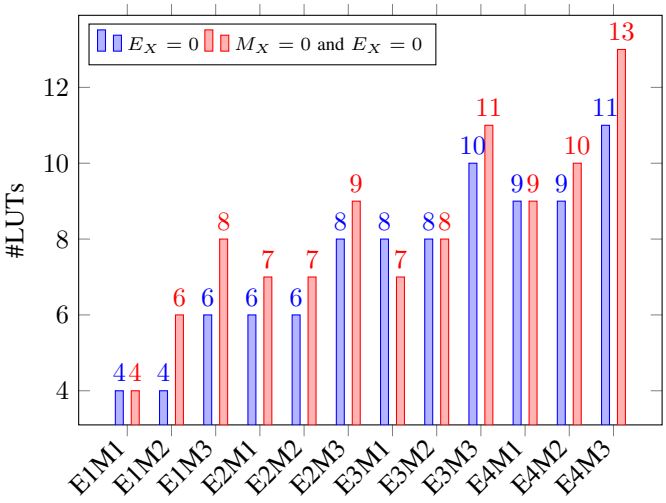
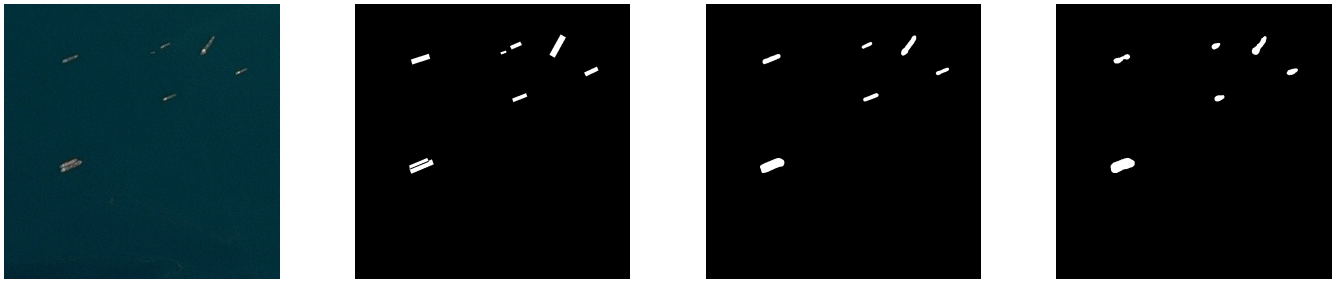


Fig. 7: LUT consumption when implementing a minifloat multiplier with two different zero encodings.

Compared to our minifloat formats where zeros are encoded with $M_X = 0$ and $E_X = 0$, the formats from [24] are slightly different, opting for a larger range of zero code words corresponding to $E_X = 0$. While the AMD-Xilinx choice

Original image | Ground truth | Single-precision | E3M2 minifloat

Fig. 3: Prediction of a single ship with original image and its associated masks.



Original image | Ground truth | Single-precision | E3M2 minifloat

Fig. 4: Prediction of multiple ships and side-by-side ships with original image and its associated masks.

Original image | Ground truth | Single-precision | E3M2 minifloat

Fig. 5: Small ships in harbor with original image and its associated masks.

leads to a slightly more efficient minifloat multiplier design (see Fig. 6 for a schematic view of the multiplier and an explanation), our synthesis results using Verilog and Vivado 2022.1 with a Zynq UltraScale+ ZCU102 as target show that the differences in LUT count between the two zero encoding choices are modest (see Fig. 7). The extra encoding space saved by our choice ($M_X \neq 0$) has a positive impact on accuracy, as can be seen Table II for the E3M2 and E3M3 formats. We believe that these properties make the $M_X = 0$ and $E_X = 0$ encoding a better choice in practice.

The AMD-Xilinx ResNet-50 implementation uses a real-valued scaling factor. With an integer exponent bias like we propose, the logic needed to handle its propagation (*e.g.* in convolution operations) would amount to a simpler integer exponent shift. In the case of a hybrid MAC design, this could also potentially lead to simpler logic when converting from integers (accumulator outputs) to minifloat (multiplier operands). We leave the testing of these statements as future work.

V. Conclusion & On-going work

Data transmission from satellites remains challenging due to the many limitations of downlink communication. While DNNs are successfully applied to spatial data processing, deep learning algorithms are now being considered as an on-board alternative to extract the relevant data to be sent to ground stations. However, strict hardware limitations of on-board systems make the use of vanilla DNN models in 32-bit floating-point arithmetic unrealistic.

We propose a QAT algorithm for learning compressed low-precision floating-point DNN models. In addition, we learn the exponent biases of each layer for both weights and activations.

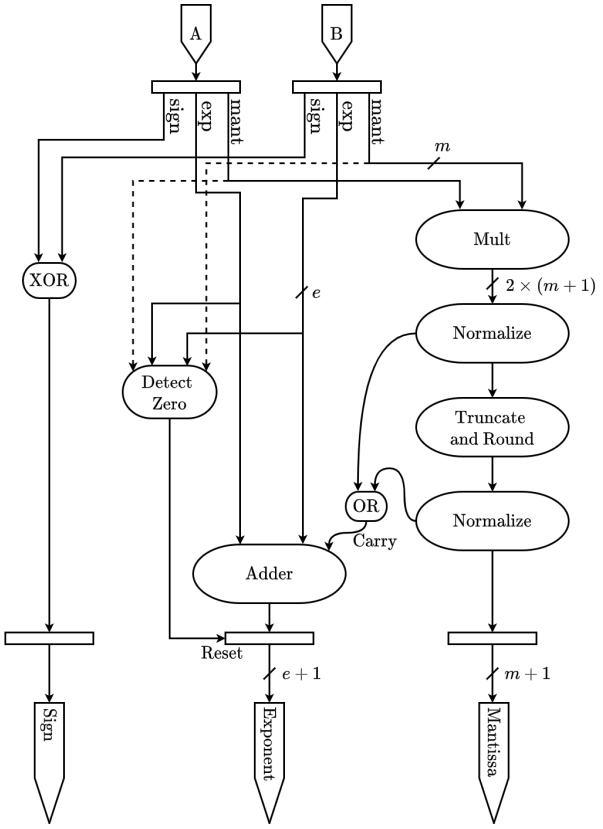


Fig. 6: Implementation of a minifloat EeMm multiplier (adapted from [24]). The minifloat multiplier preserves exponent accuracy by using a $(e+1)$-bit exponent output and truncates the mantissa output to $m+1$ bits, reducing the size of the logic needed to convert minifloat to fixed point in a hybrid minifloat fixed-point MAC design. To implement the $E_X = 0, M_X = 0$ zero encoding at the hardware level, we add the input mantissas as inputs to the block *Detect Zero* (dashed lines). This slightly complicates the zero detection logic compared to the $E_X = 0$ design from [24].

Our experiments on the CIFAR-10 and Airbus Ship datasets show good results, with low-precision floating-point models being competitive with single precision baselines.

To show the potential impact of using floating-point data formats, we have also suggested an implementation of a minifloat-enabled multiplier based on [24] that can be used as a basis for a full DNN inference accelerator for our models. The next step will be to design, test and deploy such an accelerator for the full quantized Thin U-Net 32 on FPGA targets to better gauge the feasibility of using deep learning models with low-precision FP data in an on-board space context.

Acknowledgment

We would like to thank Stéphane May for his help and advice. This work was performed using HPC resources from GENCI-IDRIS (Grant 2023-AD011013080R1).

References

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[2] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32, 2019.

[3] Y. Bengio, N. Léonard, and A. Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation, Aug. 2013.

[4] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak. LSQ+: Improving Low-Bit Quantization Through Learnable Offsets and Better Initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–697, 2020.

[5] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan. PACT: Parameterized Clipping Activation for Quantized Neural Networks, July 2018. Number: arXiv:1805.06085 arXiv:1805.06085 [cs].

[6] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53:5113–5155, 2020.

[7] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28 – NIPS 2015*, volume 2 of *NIPS'15*, pages 3123–3131. MIT Press, 2015.

[8] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to $+1$ or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[9] F. de Vieilleville, A. Lagrange, R. Ruiloba, and S. May. Towards Distillation of Deep Neural Networks for Satellite On-Board Image Segmentation. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43:1553–1559, 2020.

[10] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio. On the Automatic Exploration of Weight Sharing for Deep Neural Network Compression. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1319–1322. IEEE, 2020.

[11] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. Learned Step Size Quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[12] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, and L. Fanucci. Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities. *IEEE Aerospace and Electronic Systems Magazine*, 35(12):44–56, 2020.

[13] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A Survey of Quantization Methods for Efficient Neural Network Inference. *arXiv preprint arXiv:2103.13630*, 2021.

[14] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, June 2018.

[19] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, et al. A Study of BFLOAT16 for Deep Learning Training. *arXiv preprint arXiv:1905.12322*, 2019.

[20] A. Krizhevsky, G. Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009.

[21] A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort. FP8 Quantization: The Power of the Exponent. *Advances in Neural Information Processing Systems*, 35:14651–14662, 2022.

[22] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-Supervised Nets. In *Artificial intelligence and statistics*, pages 562–570. PMLR, 2015.

[23] F. Li, B. Zhang, and B. Liu. Ternary Weight Networks, Nov. 2016. Number: arXiv:1605.04711 arXiv:1605.04711 [cs].

[24] P. Metzgen, S. Fang, S. Pareek, B. Tian, Y. Shan, E. Delaye, and A. Sirasao. Higher Performance Neural Networks with Small Floating Point. Technical Report WP530, AMD-Xilinx, June 2021.

[25] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, et al. FP8 Formats for Deep Learning. *arXiv preprint arXiv:2209.05433*, 2022.

[26] S. Nie, Z. Jiang, H. Zhang, B. Cai, and Y. Yao. Inshore Ship Detection Based on Mask R-CNN. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 693–696. IEEE, 2018.

[27] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[28] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei. Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[29] M. Tatsumi, S.-I. Filip, C. White, O. Sentieys, and G. Lemieux. Mixing Low-Precision Formats in Multiply-Accumulate Units for DNN Training. In *2022 International Conference on Field-Programmable Technology (ICFPT)*, pages 1–9. IEEE, 2022.

[30] S. Vaze, W. Xie, and A. I. Namburete. Low-Memory CNNs Enabling Real-Time Ultrasound Segmentation Towards Mobile Deployment. *IEEE Journal of Biomedical and Health Informatics*, 24(4):1059–1069, 2020.

[31] D. Wang, A. Haytham, J. Pottenburgh, O. Saeedi, and Y. Tao. Hard Attention Net for Automatic Retinal Vessel Segmentation. *IEEE Journal of Biomedical and Health Informatics*, 24(12):3384–3396, 2020.

[32] C. Wu, M. Wang, X. Li, J. Lu, K. Wang, and L. He. Phoenix: A Low-Precision Floating-Point Quantization Oriented Architecture for Convolutional Neural Networks. *arXiv preprint arXiv:2003.02628*, 2020.

[33] S. Zhang, R. Wu, K. Xu, J. Wang, and W. Sun. R-CNN-Based Ship Detection from High Resolution Remote Sensing Imagery. *Remote Sensing*, 11(6):631, 2019.

[34] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients, Feb. 2018.