Deep PeNSieve: A deep learning framework based on the posit number system

Article *in* Digital Signal Processing · May 2020

DOI: 10.1016/j.dsp.2020.102762

3 authors:

Raul Murillo
Complutense University of Madrid
21 PUBLICATIONS **182** CITATIONS
SEE PROFILE

Alberto Del Barrio
Complutense University of Madrid
98 PUBLICATIONS **832** CITATIONS
SEE PROFILE

Guillermo Botella
Complutense University of Madrid
152 PUBLICATIONS **1,468** CITATIONS
SEE PROFILE

Deep PeNSieve: a Deep learning framework based on the Posit Number System

Raul Murillo, Alberto A. Del Barrio, Guillermo Botella

Department of Computer Architecture and Automation
Complutense University of Madrid, Madrid, Spain

Abstract

The Posit Number System (PNS) was introduced by John L. Gustafson in 2017. The interesting properties of this novel format can be exploited under the scenario of deep neural networks. In this paper, we propose Deep PeNSieve, a framework for entirely performing both training and inference of deep neural networks employing the PNS. Furthermore, an 8-bit posit quantization approach using fused operations is introduced. In comparison with the state-of-the-art posit frameworks, the proposal has been able to train more complex networks than the feedforward ones, achieving similar accuracies as the floating-point format. The case of CIFAR-10 is especially remarkable, as 16-bit posits even obtain 4% higher top-1 for such dataset. Overall, results show that the proposed quantization approach can preserve model accuracy in the same manner as common quantization techniques.

Keywords: Posit arithmetic, Deep neural network, Training, Inference, Floating-point arithmetic

---

1. Introduction

With the rising popularity of Deep Neural Networks (DNNs), a variety of digital signal applications have constructed around them. Such is the case of

Email addresses: `ramuri01@ucm.es` (Raul Murillo), `abarriog@ucm.es` (Alberto A. Del Barrio), `gbotella@ucm.es` (Guillermo Botella)

digital modulation signals [1] or speech recognition [2, 3]. In the field of computer vision, Convolutional Neural Networks (CNNs) [4, 5] have shown exceptional inference performance when trained with huge amounts of data. In fact, problems where humans used to perform better than machines, such as image classification, can now be solved with such methods, outperforming human accuracy. The IEEE 754 single-precision floating-point format [6, 7, 8] has been widely used for training DNNs. Nevertheless, floating-point formats represent real numbers uniformly, in contrast with the non-uniform distribution of DNN parameters, and therefore these formats may not be efficient in Deep Learning (DL) tasks.

The recent breakthroughs claimed by the Posit Number System (PNS), proposed by John L. Gustafson as an alternative to the IEEE 754 standard, have put this format in the spotlight. As it is illustrated in [9, 10], there are important benefits when using posits than conventional floating-point, as a better dynamic range, accuracy and closure, tapered precision and consistency between machines. This suggests that the PNS may be suitable for performing DL tasks. However, posits are still in development and there is still some controversy about their improvement [11, 12].

Researchers have recently focused on exploiting the benefits of this novel format in the DL field. However, the majority of the studies focus only on the inference phase of DNNs; they perform training with 32-bit single-precision floating-point numbers (or floats) and quantize weights and activations to low-precision posits [13, 14, 15, 16], with the consequent loss of precision [17]. Only few works explore the use of PNS for DNN training, but just in feedforward neural networks [18, 19, 20], or with the help of 32-bit floats [21]. Hence, in this paper, we propose Deep PeNSieve, a framework to directly perform the training of DNNs, as well as the inference stage, employing just the PNS. To the best of our knowledge, this is the first work proposing to do both training and inference entirely with the posits on CNNs. More concretely, the main contributions of this paper are:

1. A framework for performing both CNN training and inference stages with the PNS.
2. An approach for low-precision inference employing 8-bit posits and fused operations.
3. Results on the CIFAR-10 dataset [22] are presented, obtaining more than 4% top-1 improvement on training concerning the baseline floating-point case.
4. We compare training and inference on classical CNN models using multiple precisions for each data format. Our experiments show that training with posits can keep similar accuracy while reducing bit width to half, and predicting with 8-bit posits can achieve similar results as with 16-bit float or 8-bit integers.

The rest of this paper is organized as follows. Section 2 describes the posit number system, introducing some technical details and benefits from this format. In section 3, the state-of-the-art on posits and DNNs are described. Section 4 presents our framework. In section 5 the proposal is evaluated, comparing with equivalent floating-point formats. Finally, the concluding remarks of this work are presented in section 6.

2. The Posit Number System

The numerical value of a posit number X, whose bits are distributed as shown in Fig. 1, is given by (1):

$$X = (-1)^s * (useed)^k * 2^e * 1.f \ , \tag{1}$$

$$k = -x_{n-2} + \sum_{i=n-2}^{x_i \neq\, x_{n-2}} (-1)^{1-x_i} \ , \tag{2}$$

where $useed = 2^{2^{es}}$, k is the *regime* value, e is the unsigned exponent (if $es > 0$), and f is the mantissa of the number without the implicit one. Therefore, we use the notation $\langle n, es\rangle$ to denote posits of bit width n and es bits reserved for the exponent, respectively. In terms of format layout, the main differences

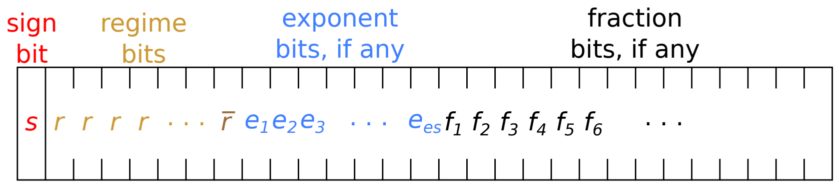


Figure 1: Layout of an $\langle n, es\rangle$ posit number. The variable-length regime field may cause exponent be encoded with less than es bits, even no bits if regime is wide enough. Same occurs with the fraction.

with floating-point are the existence of the regime field and the unsigned and unbiased exponent, if there exists such exponent field. The regime is a sequence of bits with the same value (r) finished with the negation of such value ($\overline{r}$). Provided that $X = x_{n-1}x_{n-2}...x_1x_0$, this regime can be expressed as Equation (2) shows.

In other words, the regime counts the number of occurrences of the bit labeled as r in Fig. 1. If $r = 1$ then the regime is the number of 1's minus 1, while if $r = 0$, the regime is the negative value of the number of 0's. For instance, if the regime is 4-bits wide, the value `1110` would be interpreted as $k = 2$, while the value `0001` would be $k = -3$. Hence, detecting the leading 1's and 0's is critical for performing this step. However, the main difficulty to detect the regime, and consequently unpack the posit, is that its length varies dynamically. In numbers close to zero, there will be many fraction bits and few regime bits and, in large numbers, there will be many regime bits and fewer fraction bits, but there is no fixed amount of bits.

Besides what has been mentioned, posits possess the so-called *tapered precision* [14] property, which suggests that they may be suitable for performing DL tasks. When we employ a number system with tapered precision, such as posits, the values mass around 0 following a normal distribution. That is the same distribution that the weight parameters of DNNs usually follow, but even more grouped around 0 than posit values—in concrete, they usually concentrate in the interval $[-1, 1]$. Fig. 2 illustrates this concept, which suggests that using posits for DNNs may indeed provide more accurate results than floats, whose distribution is sparser in this interval, the so-called *golden-zone* by Dinechin et al. [11]. Moreover, the posit numeric format keeps total order at bit-level, as Fig. 3 illustrates; therefore, the comparison between two posits is similar to the comparison between two integers—much easier than comparing two floating-point numbers—, which could accelerate the pooling layers of CNNs when deployed in hardware. And, what is more, for the particular case of $es = 0$, posits possess a fast approximation of the sigmoid function [9], widely employed in the DNN scenario. Finally, among all the features that this novel format has [9, 10], perhaps the most useful for the DL area is that the posit environment, according to the draft posit standard [23], mandates the implementation of fused operations. In particular, the dot product is widely used on NNs, and it can be computed as a fused operation. In the PNS, a fused operation is an operation that is not rounded until the entire expression is evaluated exactly. To avoid intermediate rounding posits make use of the so-called *quire*, a large Kulisch-like accumulator [24]. The quire has a recommended size of $n^2/2$, which is 32, 128, 512 or 2048 bits for the posit common configurations $\langle 8, 0\rangle$, $\langle 16, 1\rangle$, $\langle 32, 2\rangle$ or $\langle 64, 3\rangle$, respectively. In contrast to the Floating-Point Fused Multiply-Add (FP-FMA) [25], which was included in the first revision (2008) of the IEEE 754 standard [7], in posit arithmetic the quire unit is accessible by the programmer in terms of loading and storing data into it, or adding the result of and operation to its content, which makes this a really powerful concept.

3. Related work

Since the posit number system was introduced, the interest in scenarios where posits can outperform floats has increased rapidly. In particular, this novel format seems very promising in the field of DL. The main problematic with the PNS is that there is currently no available hardware support for this format. To the best of our knowledge, only few parameterized designs for posit arithmetic units have been proposed in the literature [26, 27, 12, 18, 28]. **It is truly interesting the approach presented in [18, 28], as authors pro-**

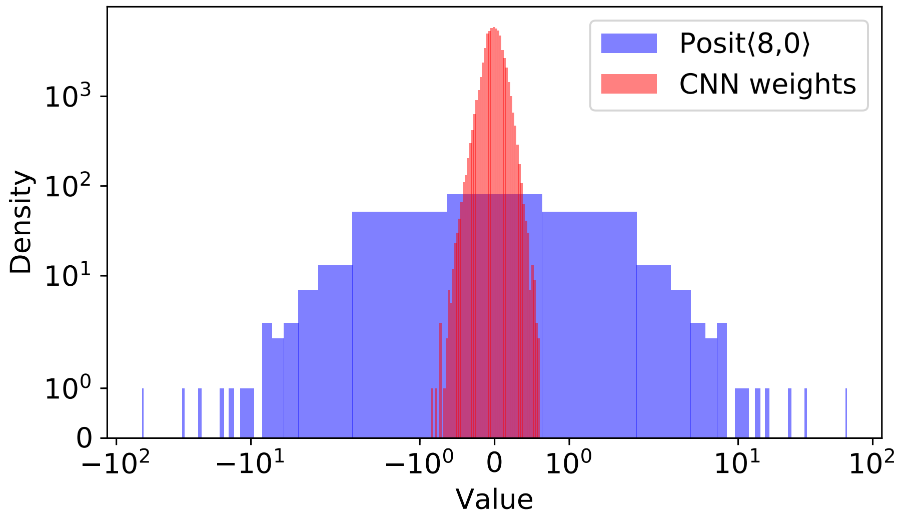
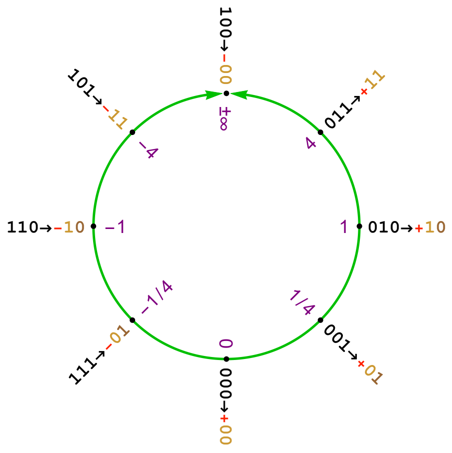
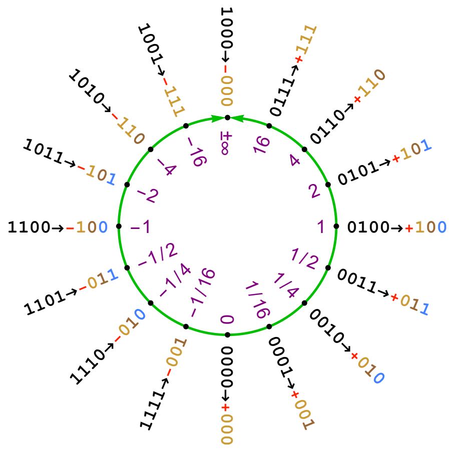


Figure 2: Histogram of posit values and trained weights of LeNet-5 on MNIST dataset. Both value sets follow normal distributions. All the CNN weights are in the range of values with higher posits density.



(a) Posit⟨3, 1⟩



(b) Posit⟨4, 1⟩

Figure 3: Visual representation in the real projective line of the posit format. (a) Posit with $n = 3$ bits and $es = 1$. (b) Posit with $n = 4$ bits and $es = 1$.

pose the use of FloPoCo, a C++ framework for the generation of arithmetic datapaths which provides a command-line interface that inputs operator specifications and outputs synthesizable VHDL [29]; this tool allows operators to be automatically generated with the specified parameters and, therefore, to obtain posit operators for arbitrary values of $\langle n, es \rangle$ with the same base design.

The majority of works using PNS perform arithmetic operations via software emulation. In [13] J. Johnson employs posit addition to complement the logarithmic multiplication [30, 31, 32, 33] when performing inference in CNNs, but just in simulation. Other studies tackling the inference stage of CNNs are performed in [14, 18, 15, 16]. For the inference stage with low-precision posits, Carmichael et al. [15, 16] use a DNN accelerator with fused multiply-and-accumulate units. In all these works, the training stage is performed in floating-point, while the inference stage is performed in low-precision posit format. In order to avoid this conversion and show the whole potential of posits [17], it would be desirable to perform training using posit format. To the best of our knowledge, the recent works in [18, 19, 20, 21] are the only ones entirely performing this phase with the PNS. Nonetheless, [21] relies on 32-bit floats for initial stages of training, while the others just deal with feedforward neural networks, which are simpler than CNNs.

Although there are clear use cases for the posit system, floating-point still possesses certain advantages [11]. The cost of posit hardware and the lack of tools in the posit landscape makes it difficult to assess the progress of the posits. Therefore, in this paper, we present Deep PeNSieve, a framework to entirely train DNNs using the PNS during the whole process, and besides, to perform low-precision inference with 8-bit posits. Furthermore, the CIFAR-10 dataset [22] will be evaluated as well, which is more complex than the datasets studied in the aforementioned works.

4. Deep PeNSieve

In this work, we propose a novel posit-based framework for CNNs. Deep PeNSieve performs the whole training stage employing the posit format. Trained models are saved preserving the format in order to perform inference with the same or even lower precision. As has been mentioned before, posits possess interesting properties that may be exploited particularly in the DNNs domain. Let us now describe in detail how training and inference are performed.

4.1. DNN training with posits

The training flow of the proposed framework is depicted in Fig. 4. In the first place, it must be noted that the inputs of networks must be in posit format and, therefore, the outputs will be in such format too. When creating a new DNN—which may include convolutional and fully-connected layers, but not only—all parameters are initialized employing the posit format. To train the model, in our case using the Adam method [34], the optimizer parameters must be also converted to the posit format. Training models are implemented with the TensorFlow [35] framework. The posit number format is extended to this framework via software emulation. At the time the network is generated using only posits with the selected $\langle n, es\rangle$ configuration, it is trained as usual, with the single difference that all computations are performed with posit numbers, which allows us to evaluate the true performance of such format in this task. In this manner, the trained parameters and models are saved to perform the inference stage with the same or lower precision.

Finally, it is interesting to point that Deep PeNSieve performs posit number computations via software emulation, which provokes an overhead regarding the baseline single-precision floating-point (Float 32) training cases. Therefore, comparing training times between different numeric formats is beyond the scope of this work, as the lack of dedicated hardware is a further handicap for the PNS.

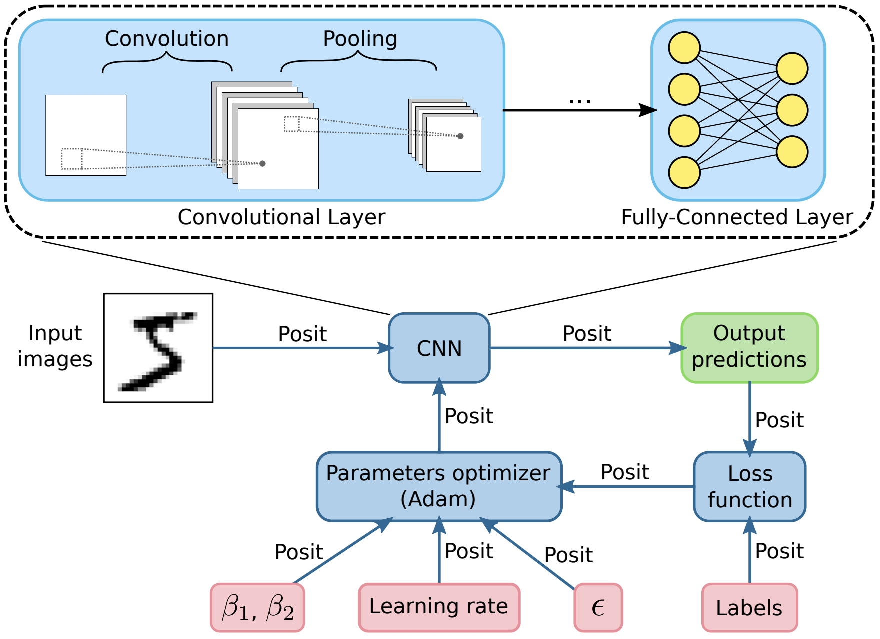


Figure 4: Illustration of the training flow of the proposed framework. Red boxes represent input data, including hyperparameters, while the blue boxes correspond with the functionalities performed by the TensorFlow framework.

4.2. DNN inference with low-precision posits

Post-training quantization is a widely used conversion technique consisting of storing tensors at lower bit widths and which can reduce the model size while also improving latency, with little degradation in model accuracy. Typical lower-precision numerical formats used in quantization are 8-bit integers (INT8), half-precision floating-point format (Float 16) and brain floating-point format (bfloat16 or BF16). Nonetheless, the majority of literature on neural network quantization involves either training from scratch [36, 37] or fine-tuning the pre-trained models [38, 39].

Deep PeNSieve allows post-training Posit$\langle 8, 0\rangle$ quantization. The stored model can be converted to a low-precision one by keeping the model architecture and converting the original parameters to Posit$\langle 8, 0\rangle$ format so that the operations of the model are virtually not changed, only the data format. The proposed solution consists in performing a linear quantization of model parameters to lower posit precision format as (3) shows.

$$\tilde{x} = round(clip(x, minpos, maxpos)) \ , \tag{3}$$

where x is the original value, $round()$ performs rounding according to selected precision, $minpos = useed^{2-n}$ is the smallest nonzero value expressible as a posit in such precision, $maxpos = useed^{n-2}$ is, analogously, the largest real value expressible as a posit and function $clip$ is defined by (4).

$$clip(x, minpos, maxpos) = \begin{cases} sign(x) \times maxpos & \text{if } |x| > maxpos \\ sign(x) \times minpos & \text{if } |x| < minpos \\ x & \text{otherwise} \end{cases} \tag{4}$$

It is worthy to note that posit arithmetic does not underflow or overflow, and clipping posits should be implicitly made by either hardware or software. Eventually, note that it is not needed to quantize activations since only precision is reduced, while arithmetic is maintained. **This is an advantage in contrast with classical quantization processes: when quantizing floating-point**

models to 8-bit precision, the dynamic range of unbounded activations, such as ReLU, needs to be calculated using calibration data [40].

4.3. Posit fused dot product in reduced precision

The current trend when quantizing a trained neural network is using INT8 as low-precision format. However, arithmetic operations in quantized neural networks using 8-bit integers requires INT16 or INT32 in practice, since INT8 can barely hold the result of certain operations such as multiplication and addition. In particular, General Matrix Multiply (GEMM), which is the core operation of NNs, involves lots of multiplications and additions and many deep learning frameworks use 32-bit accumulators for intermediate results [38]. Furthermore, novel techniques for training DNNs with low-precision arithmetic [41, 42] make use of higher (16 or 32) bit width accumulators to maintain model accuracy across GEMM functions during forward and backward passes. This suggests that quantized 8-bit posit models would require an additional structure to preserve accuracy. As it has been mentioned, the quire register is an accumulator included in the draft posit standard [23] specially designed for fused operations. Therefore, it would be interesting to use the quire for the fused dot product operations of GEMM for both convolutional and fully connected layers of 8-bit posit neural networks.

For that purpose, the internal structure of such layers must be modified, so that the GEMM function is performed with the fused dot product. Although the TensorFlow framework provides useful tools for designing and training NNs, posit fused operations are unsupported at the moment of writing. The SoftPosit[1] reference library, however, includes posit fused operations employing quire accumulator. It is necessary to re-implement the same network architecture with this library accordingly for post-training quantization using Posit$\langle 8, 0\rangle$ with quire. In particular, for an 8-bit length posit, the draft standard specifies

[1]https://gitlab.com/cerlane/SoftPosit

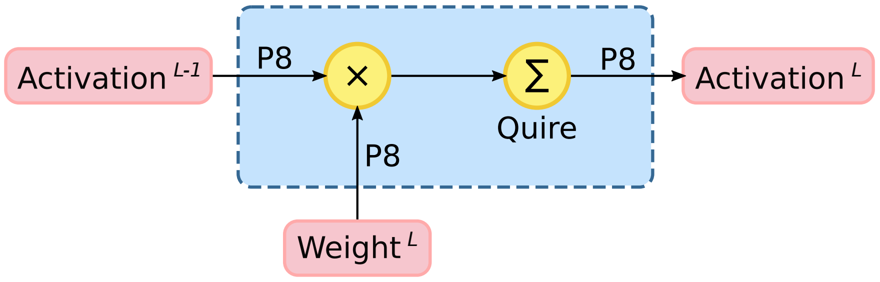


Figure 5: Low-precision data flow for GEMM function in forward propagation at layer L. Multiplications are performed without rounding. The corresponding quire size for Posit$\langle 8, 0\rangle$ is 32 bits.

that the size of the quire is 32 bits. Fig. 5 shows how the GEMM functions are implemented using the quire during the forward stage **and, for the sake of completeness, Algorithm 1 describes this kernel modification using a single quire accumulator. Note how the intermediate results are stored in a quire, so just one rounding is performed at the end of each computation.**

Algorithm 1 GEMM function with quire

```
Require: $X \in \mathcal{M}_{n\times l},\ W \in \mathcal{M}_{l\times m}$, in posit format
Ensure: $Y = X \times W \in \mathcal{M}_{n\times m}$
1: for all $i \in [0, n]$ do
2:     for all $j \in [0, m]$ do
3:         $q \leftarrow quire(0)$
4:         for all $k \in [0, l]$ do
5:             $q \leftarrow x_{i,k} \cdot w_{k,j} + q$     // The result is accumulated, but not rounded
6:         end for
7:         $y_{i,j} \leftarrow posit(q)$     // The operation is rounded here
8:     end for
9: end for
```

The GEMM function is inherited in fully connected layers, but not so for the convolutions. The convolutional layers are the most computationally intense parts of CNNs. Currently, the common approach to implement convolutional layers is to expand the image into a column matrix (im2col) and the convolution kernels into a row matrix. This way, convolutions can be performed as matrix multiplications. In this paper, we exploit this approach to benefit from the quire-based implementation of GEMM in convolutional layers.

5. Experiments and results

In this section, training and inference results of several CNN architectures, datasets, numeric formats and precision are presented. As in most DL frameworks, we use IEEE 754 single-precision floating-point format for training, while half-precision floats or 8-bit integers are used for low-precision inference. With regard to PNS, J. Gustafson recommends the use of Posit$\langle 32, 2\rangle$ and Posit$\langle 16, 1\rangle$ configurations, arguing that such 16-bit posits could be used to replace 32-bit floats [10]. Therefore, training CNNs with the aforementioned posit configurations will be compared with baseline Float 32. **Note that many deep learning frameworks use by default 32-bit floats for training (some support the so-called mixed-precision training [37]), while half-precision or 8-bit integers can be used just for inference (this is the case, for example, of PyTorch or TensorFlow, which is used in this paper).** Deep PeNSieve is employed to handle the training process with posits and TensorFlow for the rest of the formats.

With regard to Posit$\langle 8, 0\rangle$ configuration, we observed at the training stage the same behavior as previous works [18] show—networks with such configuration do not converge. For this reason, CNN training on Posit$\langle 8, 0\rangle$ is not included in this work. Nonetheless, accuracy results when performing post-training quantization from Posit$\langle 32, 2\rangle$ to Posit$\langle 8, 0\rangle$ (with and without the use of quire and fused operations) are compared to conventional Float 32-based quantization techniques.

Experiments were run on a computer with an Intel Core i7-9700K processor running at 3.60 GHz using 32 MB of RAM, running Ubuntu

18.04.

5.1. Benchmarks

To compare the benefit of posits for DL tasks, two distinct CNNs architectures have been employed. **In first place, classical LeNet-5 [4] architecture is tested with two 28×28 greyscale image datasets - MNIST [4] and Fashion-MNIST [43]. LeNet-5 consists on two convolutional layers using filters of size 5×5 and a stride of one, alternated with two pooling layers with filter size of 2×2 and a stride of two, followed by two fully connected layers and the output layer. This makes a total of 61,706 trainable parameters. In this paper, Max Pooling has been employed instead of Average Pooling as in the original paper, as well as ReLU instead of hyperbolic tangent activation, since this approach is more frequently used. The second architecture used to test the framework is a variant of CifarNet [44], a convolutional network designed to solve the CIFAR-10 classification problem. Although the architecture of this network (depicted in Fig. 6) is similar to LeNet-5, the hidden layers contain much more feature maps for handling color images, which raises the amount of trainable parameters up to 1,756,426. SVHN [45] and CIFAR-10 [22] are the two 32×32 RGB image datasets employed with this network.**

5.2. DNN training results

The aforementioned architectures are implemented and trained with three different formats: Float 32 (default precision for training), Posit$\langle 32, 2\rangle$ and Posit$\langle 16, 1\rangle$. All the models are trained by using Adam optimizer, with a learning rate of 0.001 and a batch size of 128, along 30 epochs. We do not use regularization techniques such as normalization and dropout. Input data are normalized into the range $[-1, 1]$. Table 1 shows comparisons among the inference results of standard Float 32 and posits on different trained models. As can be seen, models trained with PNS present similar results than the baseline

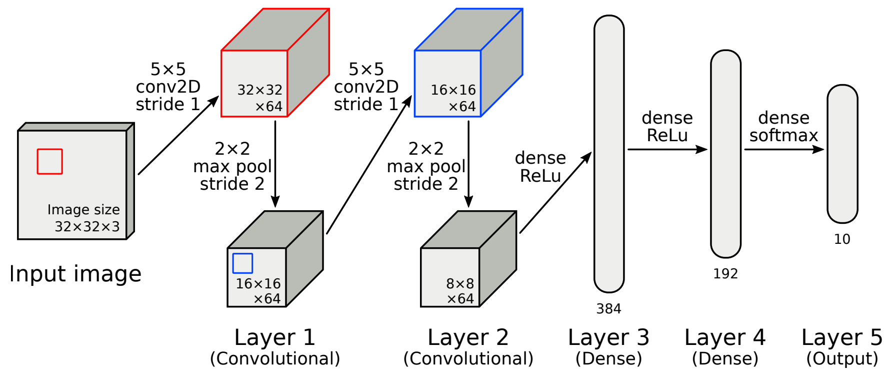


Figure 6: Illustration of the CifarNet, with $\sim 1.7 \cdot 10^6$ parameters. Convolutional filters are 5×5 with stride 1. Pooling filters are 2×2 with stride 2. ReLu is used as activation at hidden layers.

Float 32. Moreover, networks employing Posit$\langle 16, 1\rangle$ show higher accuracy than 32-bit formats—especially for the complex CIFAR-10 dataset, where Top-1 is more than 4% higher than that obtained with Float 32. While this is a very significant improvement, it does not mean yet that posits behave better than FP. For the sake of completeness, regularization techniques should be ported to the PNS as well, although this escapes the scope of this work.

Besides the accuracy, it is important to note the model size reduction of low precision formats: while 32-bit models require 724 KB and 21 MB when stored on disk for LeNet-5 and CifarNet, respectively, models on Posit$\langle 16, 1\rangle$ format require just 362 KB and 10.5 MB, respectively. Thus, with the corresponding posit hardware support the use of 16-bits posits would reduce the memory usage of the NNs, enabling training larger models or with larger mini-batches [37].

Figures 7, 8, 9 and 10 illustrate the learning processes of CNNs with different number formats on MNIST, Fashion-MNIST, SVHN and CIFAR-10 datasets, respectively. The CNNs implemented on the PNS converge in a similar manner as the floating-point ones. Nonetheless, networks on Posit$\langle 16, 1\rangle$ present lower error throughout training than other formats, which leads to higher accuracy results as mentioned before.

Table 1: Accuracy results for the inference stage

	MNIST		Fashion-MNIST		SVHN		CIFAR-10	
Format	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Float 32	99.17%	100%	89.34%	99.78%	89.32%	98.35%	68.06%	95.15%
Posit$\langle 32, 2\rangle$	99.09%	99.98%	89.90%	99.84%	89.51%	98.36%	69.32%	96.59%
Posit$\langle 16, 1\rangle$	99.18%	100%	90.17%	99.81%	90.90%	98.72%	72.51%	97.40%

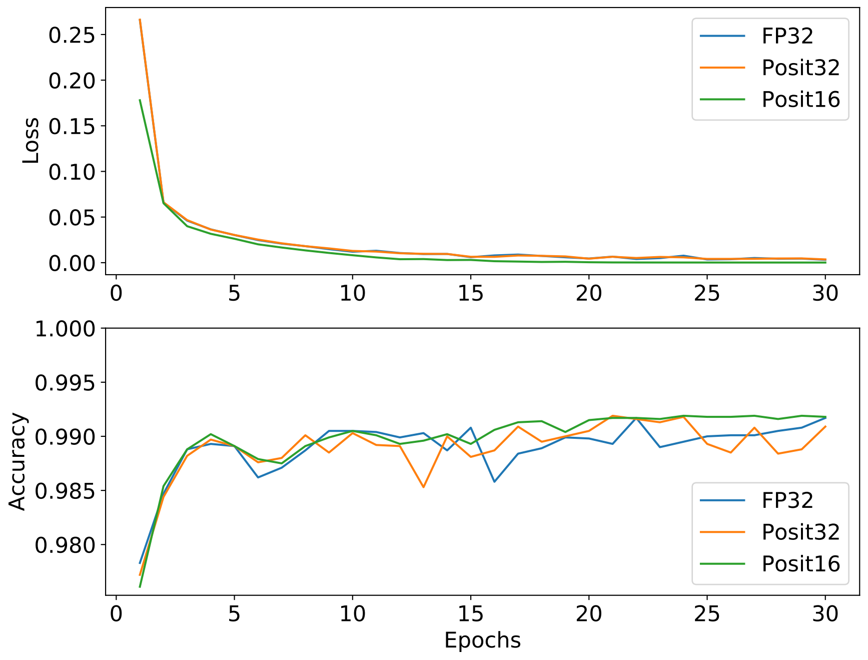


Figure 7: Learning process along LeNet-5 training on MNIST. Train loss and Top-1 validation accuracy per epoch.

5.3. DNN post-training quantization results

The previously trained models are kept **unchanged** in order to perform low-precision inference. We compare common NN quantization methods for Float 32 and Posit$\langle 32, 2\rangle$ models. For the floating-point case, Float 16 quantization and integer quantization techniques are employed to obtain models that entirely work in Float 16 and INT8 formats, respectively. With regard to posits, the

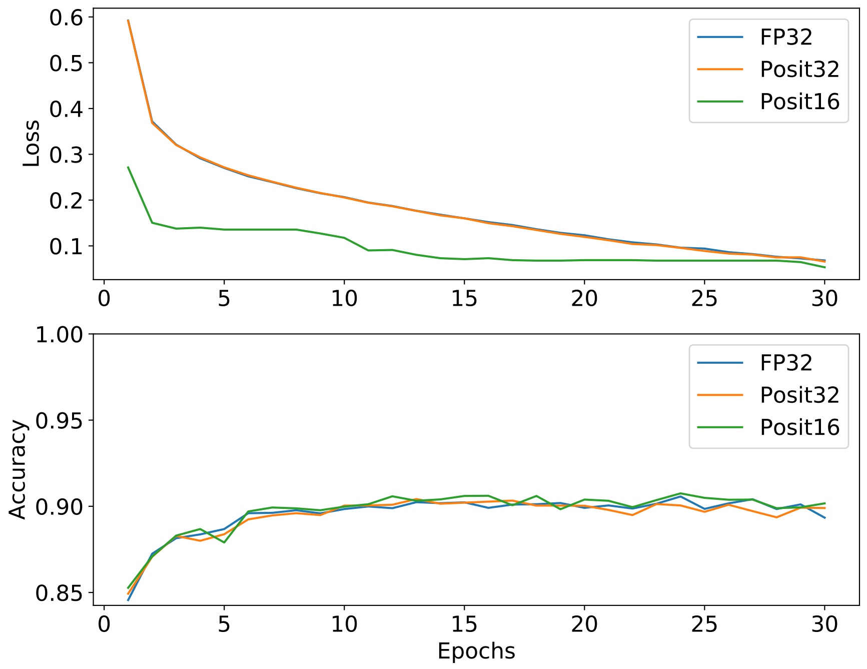


Figure 8: Learning process along LeNet-5 training on Fashion MNIST. Train loss and Top-1 validation accuracy per epoch.

32-bit models have been quantized to Posit$\langle 8, 0\rangle$, as described in Section 4.2. Moreover, this work compares inference results between naive posit quantization and employing the 8-bit posit fused dot product approach, which additionally requires changing convolutional and fully connected layers of the models as in Section 4.3.

Table 2 shows the inference results for the different post-training quantization techniques. Note that quantized 16-bit and 8-bit models are, respectively, 1/2 and 1/4 the size of the corresponding original 32-bit models. Results confirm the potential of conventional quantization techniques. Nevertheless, it is well-known that for larger networks such as MobileNet, integer quantization provides higher accuracy degradation [40]. On the other hand, there is a notable difference in the posits case between using quire and fused operations or not. Networks using fused dot product with quire get much higher accuracy (25%

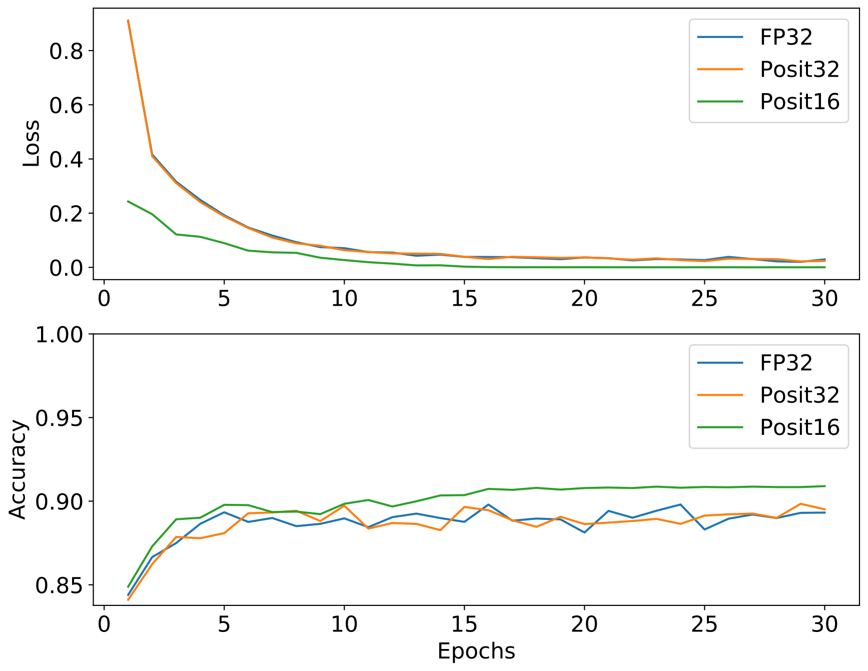


Figure 9: Learning process along CifarNet training on SVHN. Train loss and Top-1 validation accuracy per epoch.

higher top-1 for CIFAR-10) as not using, and this difference becomes more noticeable with more complex networks. In the worst case, on CIFAR-10 dataset, accuracy degradation of Posit$\langle 8, 0\rangle_{\text{quire}}$ is just 0.44% with respect the original 32-bits model, which is more than acceptable.

The proposed low-precision posit approach provides similar results as the widely used post-training quantization techniques provided by the TensorFlow framework. Higher accuracy of PNS on Fashion-MNIST and CIFAR-10 datasets might be the result of the fact that initial models on Posit$\langle 32, 2\rangle$ format present higher results than corresponding Float 32 models as well.

6. Conclusions

In this paper, Deep PeNSieve has been presented, which is a framework for performing both training and inference on DNNs entirely using the posit number

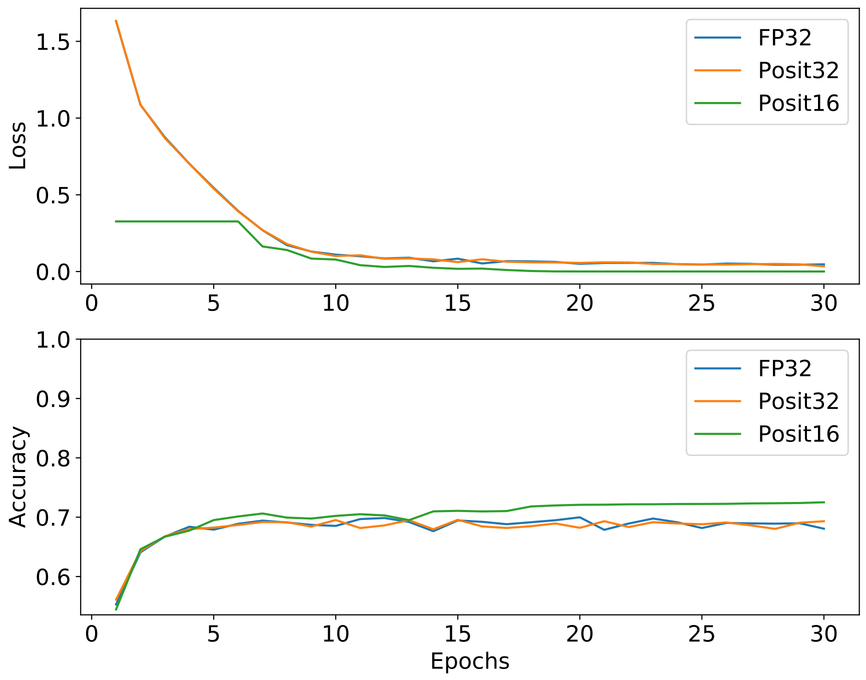


Figure 10: Learning process along CifarNet training on CIFAR-10. Train loss and Top-1 validation accuracy per epoch.

system. Deep PeNSieve allows training with Posit$\langle 32, 2\rangle$ and Posit$\langle 16, 1\rangle$, as well as performing post-quantization to Posit$\langle 8, 0\rangle$ with the support of the quire and the fused dot product. The major novelty of this work relies on performing the whole training with posits and evaluating this on CNNs, outperforming the state-of-the-art approaches, which only achieved training on smaller feedforward networks. Experiments reveal that directly training with posits gets better results than training with floating-point and quantizing to posits. Additionally, the proposed posit quantization with fused dot product outperforms naive posit quantization presented in previous works. This is especially remarkable in the case of CIFAR-10, where we can mitigate a 25% top-1 decay thanks to our approach. The empirical results demonstrate that 8-bit posit quantization can preserve model accuracy in the same manner as 8-bit integer quantization, even without re-adjusting activations or modifying the original models.

Table 2: Post-training quantization accuracy results for the inference stage

	MNIST		Fashion-MNIST		SVHN		CIFAR-10	
Format	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Float 16	99.17%	100%	89.34%	99.78%	89.32%	98.35%	68.05%	96.15%
INT8	99.16%	100%	89.51%	99.79%	89.33%	98.38%	68.15%	96.14%
Posit$\langle 8, 0\rangle$	98.77%	99.99%	88.52%	99.82%	81.31%	97.07%	43.89%	86.49%
Posit$\langle 8, 0\rangle_{\text{quire}}$	99.07%	99.99%	89.92%	99.81%	89.13%	98.39%	68.88%	96.47%

The results in this paper are promising, like this 4% top-1 increase when using Posit$\langle 16, 1\rangle$ for training on CIFAR-10. Nevertheless, future work should tackle the adaptation to the posit format of optimization techniques during training as the batch normalization, which could provide a more insightful view of the improvements achieved by Deep PeNSieve.

Appendix

The source code of the proposed framework, Deep PeNSieve, will be provided as an open-source software and will be available at `https://github.com/RaulMurillo/deep-pensieve`. This open-source proposal on posit arithmetic would provide a basic platform for more advances and research on posit arithmetic and its application on deep learning.

Acknowledgments

This paper has been supported by the CM under grant S2018/TCS-4423, the EU (FEDER) and the Spanish MINECO under grant RTI2018-093684-B-I00 and by Banco Santander under grant PR26/16-20B-1.

References

[1] A. Ali, F. Yangyu, S. Liu, Automatic modulation classification of digital modulation signals with stacked autoencoders, Digital Signal Processing 71 (2017) 108–116. doi:https://doi.org/10.1016/j.dsp.2017.09.005.

[2] O. Abdel-Hamid et al., Convolutional neural networks for speech recognition, IEEE/ACM Transactions on Audio, Speech, and Language Processing 22 (10) (2014) 1533–1545. doi:10.1109/TASLP.2014.2339736.

[3] M. Lee, J. Lee, J.-H. Chang, Ensemble of jointly trained deep neural network-based acoustic models for reverberant speech recognition, Digital Signal Processing 85 (2019) 1–9. doi:https://doi.org/10.1016/j.dsp.2018.11.005.

[4] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. doi:10.1109/5.726791.

[5] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems 25, 2012, pp. 1097–1105.

[6] IEEE standard for binary floating-point arithmetic, ANSI/IEEE Std 754-1985 (1985) 1–20doi:10.1109/IEEESTD.1985.82928.

[7] IEEE standard for floating-point arithmetic, IEEE Std 754-2008 (2008) 1–70doi:10.1109/IEEESTD.2008.4610935.

[8] IEEE standard for floating-point arithmetic, IEEE Std 754-2019 (Revision of IEEE 754-2008) (2019) 1–84doi:10.1109/IEEESTD.2019.8766229.

[9] J. Gustafson, I. Yonemoto, Beating floating point at its own game: Posit arithmetic, Supercomputing Frontiers and Innovations 4 (2) (2017) 71–86. doi:10.14529/jsfi170206.

[10] J. Gustafson, Posit arithmetic, https://posithub.org/docs/Posits4.pdf, [Online; accessed 5-November-2019] (2017).

[11] F. de Dinechin, L. Forget, J.-M. Muller, Y. Uguen, Posits: The good, the bad and the ugly, in: Proceedings of the Conference for Next Generation Arithmetic 2019, CoNGA'19, ACM Press, 2019, pp. 6:1–6:10. doi:10.1145/3316279.3316285.

[12] Y. Uguen, L. Forget, F. de Dinechin, Evaluating the hardware cost of the posit number system, in: 2019 29th International Conference on Field-Programmable Logic and Applications (FPL), IEEE, Barcelona, Spain, 2019, pp. 1–8. doi:10.1109/FPL.2019.00026.

[13] J. Johnson, Rethinking floating point for deep learning, arXiv preprint arXiv:1811.01721.

[14] H. F. Langroudi, Z. Carmichael, J. Gustafson, D. Kudithipudi, PositNN framework: Tapered precision deep learning inference for the edge, in: 2019 IEEE Space Computing Conference (SCC), IEEE, 2019, pp. 53–59. doi:10.1109/spacecomp.2019.00011.

[15] Z. Carmichael et al., Deep positron: A deep neural network using the posit number system, in: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2019, pp. 1421–1426. doi:10.23919/date.2019.8715262.

[16] Z. Carmichael et al., Performance-efficiency trade-off of low-precision numerical formats in deep neural networks, in: Proceedings of the Conference for Next Generation Arithmetic 2019 on - CoNGA'19, ACM Press, 2019, pp. 3:1–3:9. doi:10.1145/3316279.3316282.

[17] S. Tiwari, N. Gala, C. Rebeiro, V. Kamakoti, PERI: A posit enabled RISC-V core, arXiv preprint arXiv:1908.01466.

[18] R. M. Montero, A. A. Del Barrio, G. Botella, Template-based posit multiplication for training and inferring in neural networks, arXiv preprint arXiv:1907.04091.

[19] H. F. Langroudi, Z. Carmichael, D. Kudithipudi, Deep learning training on the edge with low-precision posits, arXiv preprint arXiv:1907.13216.

[20] H. F. Langroudi, Z. Carmichael, D. Pastuch, D. Kudithipudi, Cheetah: Mixed low-precision hardware & software co-design framework for dnns on the edge, arXiv preprint arXiv:1908.02386.

[21] J. Lu et al., Training deep neural networks using posit number system, arXiv preprint arXiv:1909.03831.

[22] A. Krizhevsky, Learning multiple layers of features from tiny images, Master's thesis, University of Toronto, Canada (2009).

[23] Posit Working Group, Posit standard documentation, `https://posithub.org/docs/posit_standard.pdf`, [Online; accessed 5-November-2019].

[24] U. W. Kulisch, Advanced arithmetic for the digital computer: design of arithmetic units, Springer-Verlag, 2002. `doi:10.1007/978-3-7091-0525-2`.

[25] A. A. Del Barrio, N. Bagherzadeh, R. Hermida, Ultra-low-power adder stage design for exascale floating point units, ACM Trans. Embed. Comput. Syst. 13 (3s). `doi:10.1145/2567932`.

[26] M. K. Jaiswal, H. K. So, PACoGen: A hardware posit arithmetic core generator, IEEE Access 7 (2019) 74586–74601. `doi:10.1109/ACCESS.2019.2920936`.

[27] H. Zhang, J. He, S.-B. Ko, Efficient posit multiply-accumulate unit generator for deep learning applications, in: 2019 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2019, pp. 1–5. `doi:10.1109/iscas.2019.8702349`.

[28] R. Murillo, A. A. Del Barrio, G. Botella, Customized posit adders and multipliers using the FloPoCo core generator, in: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, to appear.

[29] F. de Dinechin, B. Pasca, Designing custom arithmetic data paths with FloPoCo, IEEE Design & Test of Computers 28 (4) (2011) 18–27. `doi:10.1109/mdt.2011.44`.

[30] M. S. Kim, A. A. D. Barrio, R. Hermida, N. Bagherzadeh, Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks, in: 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, 2018, pp. 617–622. `doi:10.1109/aspdac.2018.8297391`.

[31] M. S. Kim et al., Efficient Mitchell's approximate log multipliers for convolutional neural networks, IEEE Transactions on Computers 68 (5) (2019) 660–675. `doi:10.1109/tc.2018.2880742`.

[32] L. T. Oliveira et al., Design of power-efficient FPGA convolutional cores with approximate log multiplier, in: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), 2019, pp. 203–208.

[33] H. Kim, M. S. Kim, A. A. D. Barrio, N. Bagherzadeh, A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks, in: 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH), 2019, pp. 108–111. `doi:10.1109/ARITH.2019.00029`.

[34] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[35] M. Abadi et al., TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL `https://www.tensorflow.org/`

[36] I. Hubara et al., Binarized neural networks, in: Advances in Neural Information Processing Systems 29, 2016, pp. 4107–4115.

[37] P. Micikevicius et al., Mixed precision training, in: International Conference on Learning Representations, (ICLR), 2018.

[38] B. Jacob et al., Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: 2018 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR), 2018, pp. 2704–2713. `doi:10.1109/CVPR.2018.00286`.

[39] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, in: 4th International Conference on Learning Representations, ICLR, 2016.

[40] R. Krishnamoorthi, Quantizing deep convolutional networks for efficient inference: A whitepaper, arXiv preprint arXiv:1806.08342.

[41] N. Mellempudi, S. Srinivasan, D. Das, B. Kaul, Mixed precision training with 8-bit floating point, arXiv preprint arXiv:1905.12334.

[42] N. Wang et al., Training deep neural networks with 8-bit floating point numbers, in: Advances in Neural Information Processing Systems 31, 2018, pp. 7686–7695.

[43] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747.

[44] J. H. Hosang, M. Omran, R. Benenson, B. Schiele, Taking a deeper look at pedestrians, in: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR), IEEE Computer Society, 2015, pp. 4073–4082.

[45] Y. Netzer et al., Reading digits in natural images with unsupervised feature learning, NIPS Workshop on Deep Learning and Unsupervised Feature Learning.