


# Reducing Memory and Computational Cost for Deep Neural Network Training with Quantized Parameter Updates


**Leo Buron**

(University of Duisburg-Essen, Duisburg, Germany)

 <https://orcid.org/0009-0001-8939-4784>, [leo.buron@uni-due.de](mailto:leo.buron@uni-due.de)


**Andreas Erbslöh**

(University of Duisburg-Essen, Duisburg, Germany)

 <https://orcid.org/0000-0001-6702-892X>, [andreas.erbsloeh@uni-due.de](mailto:andreas.erbsloeh@uni-due.de)

**Gregor Schiele**

(University of Duisburg-Essen, Duisburg, Germany)

 <https://orcid.org/0000-0003-4266-4828>, [gregor.schiele@uni-due.de](mailto:gregor.schiele@uni-due.de)

**Abstract:** For embedded devices, both memory and computational efficiency are essential due to their constrained resources. However, neural network training remains both computation and memory intensive. Although many existing studies apply quantization schemes to mitigate memory overhead, they often employ stochastic rounding for both inference and gradient computation. Notably, no prior work has explored its advantages exclusively in parameter updates. Here, we introduce Quantized Parameter Updates (*QPU*), which uses stochastic rounding (*SQPU*) to achieve improved and more stable training outcomes. Our fixed-point quantization scheme quantizes parameters (weights and biases) upon model initialization, conducts high-precision gradient computations during training, and applies stochastically quantized updates thereafter. This approach substantially lowers memory usage and enables mostly quantized inference, thereby accelerating calculations. Furthermore, storing quantized inputs for gradient computation reduces memory demands even more. When tested on the FASHION-MNIST dataset, our method matches the Straight-Through Estimator (*STE*) in performance, delivering 0.92% validation accuracy while consuming just 57% of the memory during training. Accepting a slight 1.5% drop in accuracy yields a 50% memory reduction. Additional techniques include stochastic rounding in inference, the use of higher precision for parameters than for layer outputs to limit overflow, L2 regularization via weight decay, and adaptive learning-rate scheduling for improved optimization across a range of batch sizes.

**Keywords:** deep learning, quantized parameters, edge training, embedded deep learning, memory reduction, stochastic rounding, fixed-point quantization

**Categories:** C.1.3, D.4.7, C.3, C.5.3, I.2.6

**DOI:** 10.3897/jucs.164737

## 1 Introduction

In recent years, deep neural networks have become increasingly successful, enabling them to tackle more complex tasks. However, these successes have also led to larger and more complicated networks, making training difficult even on GPU clusters where

memory limitations can be a major hurdle [Guo et al. 2019]. This issue is even more challenging for edge and embedded computing, where resources are purposely limited. At the same time, there's a rising demand for local computing on these limited-resource devices. This demand is driven by applications like brain-computer interfaces [Grahn et al. 2014, Im and Kim 2020, Kathe et al. 2022] and human activity recognition using personal devices such as smartphones and smartwatches [Li et al. 2014, Sun et al. 2018]. These applications raise the need for data privacy, network independence and reduced transmission energy, and low latency not only for local inference, but also training of the neural network.

Traditionally, floating-point arithmetic using 32 bits (*FIP32*) has been the standard for neural network inference and training. To reduce the need for computational resources, quantization schemes have been proposed that reduce the bit width of parameters [Fox et al. 2021, Gennari do Nascimento et al. 2023, Zhou et al. 2016]. Those usually speed up training and inference and reduce memory consumption and memory bandwidth constraints. However, direct application of these schemes often degrades network performance. Techniques like the straight-through estimator (*STE*) have been developed to mitigate this performance loss during inference with quantized parameters but do not reduce bit width during training. The *FxP* quantization scheme, which includes an integer and a fractional part, is known for its efficiency in embedded devices [Jacob et al. 2018]. This scheme involves clamping values within a two's complement range and rounding to the nearest fractional digit, with round half-to-even (*HTE*) being the typical mode to prevent cumulative errors. However, precision loss during rounding remains a concern. Stochastic rounding (*SR*) offers a solution by sampling a uniform distribution to adjust values before rounding, as demonstrated in previous works [Gupta et al. 2015, Gennari do Nascimento et al. 2023, Sun et al. 2019, Yang et al. 2019, Zhou et al. 2016, Fox et al. 2021].

In this study, we explore the potential to reduce memory consumption during training through the use of *FxP* quantization for parameters, layer output and momentum in stochastic gradient descent (*SGD*) optimizers. We illustrate that employing *SR* exclusively in the optimizer achieves a performance comparable to state-of-the-art (*SOTA*) methods such as *STE*. Our hypotheses are verified for various *FxP* configurations on the FASHION-MNIST dataset [Xiao et al. 2017]. As it is more challenging than MNIST while retaining a comparable scale and format, Fashion-MNIST constitutes a suitable intermediate benchmark prior to evaluating performance on substantially larger datasets, such as CIFAR-10 or ImageNet. We believe that our methods are also applicable to those vision data sets.

Our main contributions are as follows:

- First, we present an approach for the SGD-based optimizer that allows quantized parameter updates *QPU*.
- Next, we present how memory consumption during training is reduced compared to *STE* with quantized parameters and quantized optimizer momentum.
- We explicitly show our iterative improvements for our set of experiments and describe our findings and problems in detail.

In the next chapter, we review related work. Subsequently, we outline our methodology. The fourth chapter details our experiments. Finally, we discuss our findings and propose directions for future research.

## 2 Related Work

The related work can be grouped into two areas. One focuses on optimizing the training process to produce a quantized model, while the other centers on optimizing the training of an already quantized model. In this paper, we first address the approach that aims to create a quantized model. Typically, such training occurs on a server or desktop system rather than on an embedded device, with inference then optimized for edge deployment. Consequently, the main objective is not to reduce the training overhead. Nonetheless, as model sizes continue to grow, lowering the training overhead is becoming increasingly important. Two key *SOTA* techniques are quantization-aware training and post-training quantization (*PTQ*). Since *PTQ* does not reduce memory consumption during training, we do not include it in our analysis.

The aim of quantization-aware training (*QAT*) is to mitigate errors introduced by quantizing the model during training, allowing the optimizer to account for any loss from quantization artifacts. One standard method, the straight-through estimator (*STE*), relies on a high-precision copy of the parameters [Bengio et al. 2013]. All parameters are quantized for inference, and each layer’s inputs are quantized to lessen the memory needed for gradient computation.

Recent research has also explored training with quantized parameters. These studies generally quantize both forward and backward passes to reduce memory and computation loads. Micikevicius et al. employ half-precision (16-bit) floating-point IEEE for multiplications, using *FIP32* accumulators, as noted in [Micikevicius et al. 2018]. They also maintain a *FIP32* copy of the model parameters. Gupta et al. propose 16-bit *FxP* quantization for all computations [Gupta et al. 2015], identifying *SR* as pivotal in the backward pass. Sun et al. show that, for an 8-bit float format, using four exponent bits for inference and five exponent bits for gradients is optimal [Sun et al. 2019], and they also employ *SR*. Gennari et al. and Fox et al. use block floating-point, sharing an exponent across a tensor to reduce memory usage [Gennari do Nascimento et al. 2023, Fox et al. 2021], and they too adopt *SR*. Zhou et al. implement a low-bit quantization scheme with *SR* [Zhou et al. 2016]. As discussed in [Gupta et al. 2015, Sun et al. 2019, Gennari do Nascimento et al. 2023, Zhou et al. 2016, Fox et al. 2021], quantizing the backward pass often increases overall loss, yet all these works apply *SR* to every calculation. They also hypothesize that *SR* would be equally useful in optimizer routines, though only Gupta et al. provide extensive details [Gupta et al. 2015]. To the best of our knowledge, no work has yet studied how restricting *SR* solely to optimizer updates affects quantized parameters.

## 3 Method

We describe the *FxP* quantization scheme in terms of an *FxP* configuration characterized by two parameters: the total bit count (including the sign bit), denoted as *TotalBits*, and the fractional bit count, denoted as *FracBits*. Additionally, the rounding mode for quantization can be either *HTE* or *SR*.

The specific implementation varies depending on the rounding mode. For *HTE*, the quantization follows Equation 1. In contrast, for stochastic rounding, a noise sample drawn from the uniform distribution  $\mathcal{U}$  is scaled by the number of fractional bits (see Equation 2). This noise is then added to the input prior to quantization (see Equation 3).

$$\text{quantize}_{HTE}(x) = \hat{x} = \frac{HTE(x * 2^{FracBits})}{2^{FracBits}} \quad (1)$$

$$noise = \frac{Sample_{\mathcal{U}} - 0.5}{2^{FracBits}} \quad \text{with } \mathcal{U}(0, 1) \quad (2)$$

$$quantizeSR(x) = \hat{x} = quantize(x + noise) \quad (3)$$

Stochastic quantization introduces additional overhead for each operation, but its actual cost depends on the target platform, which we have not specified. We express the cost  $C$  of stochastic rounding (SR), denoted as  $C_{SR}$ , in terms of the cost of sampling from a uniform distribution  $C_{Sample_{\mathcal{U}}}$ , the cost of a bit shift  $C_{BitShift}$  (equal to the number of fractional bits), the cost of adding  $C_{Add}$  this shift to the input signal, and the cost of applying HTE  $C_{HTE}$  in Equation 4.

$$C_{SR} = C_{Sample_{\mathcal{U}}} + C_{BitShift} * FracBits + C_{Add} + C_{HTE} \quad (4)$$

During neural network initialization, we apply the chosen  $FxP$  quantization scheme to each parameter. Consequently, the bit width of these values is reduced according to the  $FxP$  configuration. We express the memory reduction using Equation 5. In this work, we compare the memory utilization against  $FLP32$ . Hence, an  $FxP$  value with 8 total bits yields a memory reduction factor of 0.25, as it consumes only 25% of the memory that  $FLP32$  would require.

$$\begin{aligned} MemoryReduction &= \frac{MemoryConsumption(\hat{x})}{MemoryConsumption(x)} \\ &= \frac{TotalBits(\hat{x})}{TotalBits(x)} \end{aligned} \quad (5)$$

During inference, the same quantization scheme is applied to all layer outputs. When using SR in inference, costs are incurred for each rounding. For STE, this overhead increases more for inference because each parameter is quantized during inference. For the calculation of the gradients, each layer's input needs to be saved during inference. The optimizer calculates the parameter update  $\delta_{w,t}$  for each parameter  $w$  at time step  $t$ . When using stochastic gradient descent (SGD) optimizer  $\delta_{w,t}$  is computed from the gradient  $g_{w,t}$ , and the learning rate  $\gamma$  is used as a scaling factor. Furthermore, a momentum term is added to the gradient  $g_{w,t}$ . The momentum is calculated with the momentum buffer  $\delta_{w,t-1}$ , which is down-scaled by the momentum factor  $\mu$  (see equation 6).

$$\delta_{w,t} = (\mu * \delta_{w,t-1} + g_{w,t}) * \gamma \quad (6)$$

In addition, a weight decay can be used before calculating the  $\delta_{w,t}$ . When using a weight decay, the gradient  $g_{w,t}$  is increased in advance by the absolute value of  $w_t$  down-scaled by the weight decay factor  $\lambda$  (see equation 7).

$$g_{w,t} = g_{w,t} + \lambda * abs(w_t) \quad (7)$$

Each parameter  $w$  is updated by subtracting  $\delta_{w,t}$  from  $w_t$  resulting in  $w_{t+1}$  (see equation 8). To update the quantized parameters accordingly, the  $FxP$  quantization scheme must be applied to  $\delta_{w,t}$ . The quantized value of  $\delta$  is essential for updating  $\delta_{w,t}$  and is also stored in the momentum buffer, thereby reducing its memory consumption, as described in Equation 5.

$$w_{t+1} = w_t - \delta_{w,t} \quad (8)$$

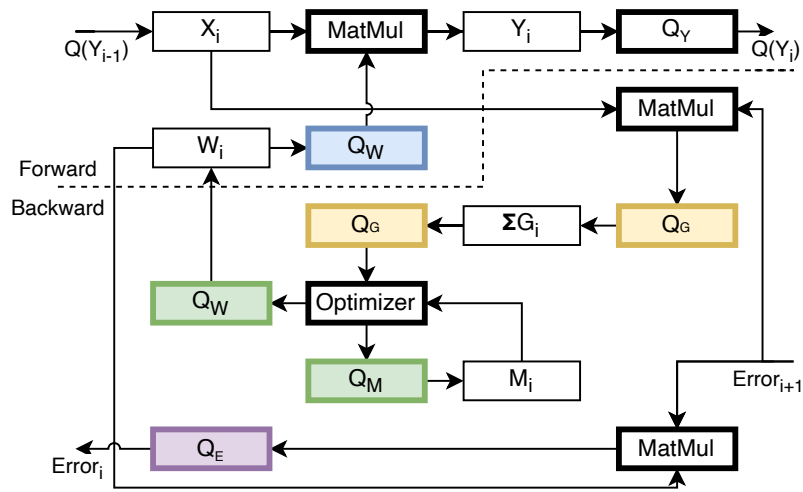


Figure 1: Dataflow during training. Quantizations only applicable for STE are marked in blue. Quantizations for SQPU are marked in green. Quantizations applicable to the methods mentioned in the related work are marked in purple

We hypothesize that employing stochastic rounding for the quantized parameter update (SQPU), with quantized  $\delta_{w,t}$  and a corresponding quantized momentum buffer  $\delta_{w,t-1}$ , achieves performance comparable to the same quantization scheme when using the quantization-aware training (QAT) method. We test this hypothesis under various FxP configurations for parameters and inference calculations, applying both stochastic rounding for the quantized parameter update SQPU and round half-to-even for the quantized parameter update (HTEQPU). In addition, we apply these FxP configurations using the QAT method.

In Figure 1, we illustrate the data flow during training for a single layer  $i$ . All boxes with a thicker border represent components that perform some operation. Quantization is indicated as  $Q$ , and it applies to the respective tensor being quantized. For all methods, the layer output  $Y_i$  is quantized. In the STE approach, the parameters  $W_i$  are quantized only when being read (marked in blue) [Bengio et al. 2013]. In contrast, we quantize the parameters  $W_i$  during their update (marked in green/purple), ensuring they remain quantized at all times and removing the need to store a full-precision copy. Moreover, we quantize the momentum  $M_i$  to reduce the required memory (marked in green). Gupta et al. additionally quantize the backpropagation of error  $Error_i$  [Gupta et al. 2015]. It is not entirely clear whether they quantize the gradient before accumulation or if they also quantize it after accumulation (marked in yellow). Note that the optimizer-related components are used only once per mini-batch to update the parameters, whereas forward and backward error propagation are computed per sample within the mini-batch.

## 4 Experiments

In this section, we summarize all our experiments. First, we assess whether our hypothesis holds for a range of  $FxP$  configurations applied to a single model architecture on the FASHION-MNIST dataset [Xiao et al. 2017]. This dataset comprises 10 categories of fashion items in 8-bit grayscale images, with 60,000 training samples and 10,000 test samples. The images are normalized to a mean of 0.1307 and a standard deviation of 0.3081. We do not quantize the input or the data, as existing work appears to adopt a similar approach but does not explicitly state it.

We then derive insights from our first experiment to design a second experiment exploring different batch sizes. Finally, in our third experiment, we introduce an improved learning-rate initialization and address stability issues for both our  $SQPU$  and  $STE$  methods.

### 4.1 Base Experiment

In this experiment we train the same model for a few  $FxP$  configurations once with  $HTE$  and once with  $SR$  during inference. We compare them with  $STE$ ,  $SQPU$  and  $HTEQP$ . Our hypothesis is that  $QP$  achieve comparable performance to  $STE$ . In addition, we believe that  $SQPU$  performs better than  $HTEQP$ . Next, we describe the model implementation and the rest of the experiment setup in detail.

Layer	Kernel	Channels	Filters	Feat-In	Feat-Out	Inputs	Round	Parameters
Conv2d	5x5	1	32	28x28	24x24	784	18432	800
MaxPool2d	2x2	32	32	24x24	12x12	18432	-	-
Conv2d	5x5	32	64	12x12	8x8	4608	4096	51200
MaxPool2d	2x2	64	64	8x8	4x4	4096	-	-
Linear	-	1	1	1024	128	1024	128	131200
Linear	-	1	1	128	10	128	10	1290
Softmax	-	1	1	10	10	10	-	-
					<b>SUM</b>	<b>29082</b>	<b>22666</b>	<b>184490</b>

Table 1: Model architecture derived from [Gupta et al. 2015]

To evaluate the stochastic update, we train a convolutional neural network (see Table 1) originally used in [Gupta et al. 2015] for the MNIST dataset [Bottou et al. 1994]. All convolutions are implemented with no bias, no padding, a dilation of one, and a stride of one. Each linear layer uses a bias. Except for the final layer, every linear and convolutional layer applies the ReLU activation function; the final layer employs the softmax activation function. We implement the quantization scheme as fake quantization. We run the experiment once in full precision ( $FP32$ ) to establish a baseline. In all other configurations, we quantize each learnable parameter and the output of every convolutional and linear layer using the same  $FxP$  setting. The output of the softmax layer remains unquantized. For the forward pass, we run each  $FxP$  configuration twice: once with  $HTE$  and once with  $SR$ . We use the cross-entropy loss function and an SGD optimizer with momentum but without weight decay. Following a small grid search with the baseline model, we set the learning rate to 0.01 and the momentum to 0.9, matching

the values in [Gupta et al. 2015]. We apply the same learning rate and momentum for all subsequent experiments and do not use a learning rate scheduler. We run each parameter update for each *FxP* configuration with the *STE* method, *SQPU*, and *HTEQPU*, all using the same five random seeds. This ensures identical model initialization and facilitates a direct comparison among the parameter update methods. Each training run lasts 50 epochs. We do not quantize the loss functions, gradients, or optimizer calculations at any point. All experiments (including those described in subsequent sections) are implemented with the PyTorch framework and executed on an RTX8000 GPU.

In Table 2 we show all 13 fixed-point configurations for this experiment. Overall this leads to 390 training runs.

Parameter / Inference Total Bits	32	32	32	24	24	24	16	16	12	12	8	8
Parameter / Inference Fractional Bits	24	20	16	16	14	12	10	8	8	6	5	4

Table 2: *FxP* configurations for Parameters and Inference Quantization used in this Experiment.

Next, we evaluate the memory consumption, the computational overhead, and the models’ accuracy. We also present the best model from the best training run and the best model from the worst training run, referring to these as the “best” and “worst” models, respectively.

The memory footprint of the model parameters scales linearly with the total number of bits. For example, a 32-bit configuration requires 737.96 kB, whereas an 8-bit configuration requires 184.49 kB. In Figure 2, we show the required memory for five different total bit widths. The memory reduction, compared to *STE*, depends on the batch size: with a batch size of 1024, the overall memory usage is dominated by the inputs required for the backward pass. When using smaller batch sizes, the bit width becomes more significant. We also note that the input to the neural network remains unquantized, resulting in the affine relationship between batch size and stored input. Consequently, in this work, we express memory reduction as a multiplicative factor, usually in the range  $[0, 1]$ .

Table 3 shows the overhead for each parameter update method. This overhead depends heavily on the rounding mode used during inference. Specifically, *STE* requires rounding of each parameter and layer output, whereas training with quantized parameters only rounds the layer outputs during inference. However, training with quantized parameters also requires rounding at each parameter update. When using stochastic rounding (*SR*) during inference, both the *STE* and *SQPU* parameter update methods perform the same number of rounding operations. Additionally, *STE* applies an up-scaling prior to rounding and a down-scaling afterward. For a batch size of 1024, this leads to a factor of nine more rounding operations for *STE* than for our parameter update method. Notably, rounding-based methods, especially stochastic rounding in *STE* and our quantized parameter update, can cause a substantial overhead.

After comparing the memory reduction and overhead, we next compare the models’ worst and best accuracies. The baseline model without quantization attains a worst accuracy of 76% and a best accuracy of 90%, with a standard deviation of 7%. We visualize a heatmap of the worst and best validation accuracies for all evaluated *FxP* configurations using round *HTE* during inference, separated by parameter update method, in Figure 3. The same visualization for *FxP* configurations with *SR* during inference is

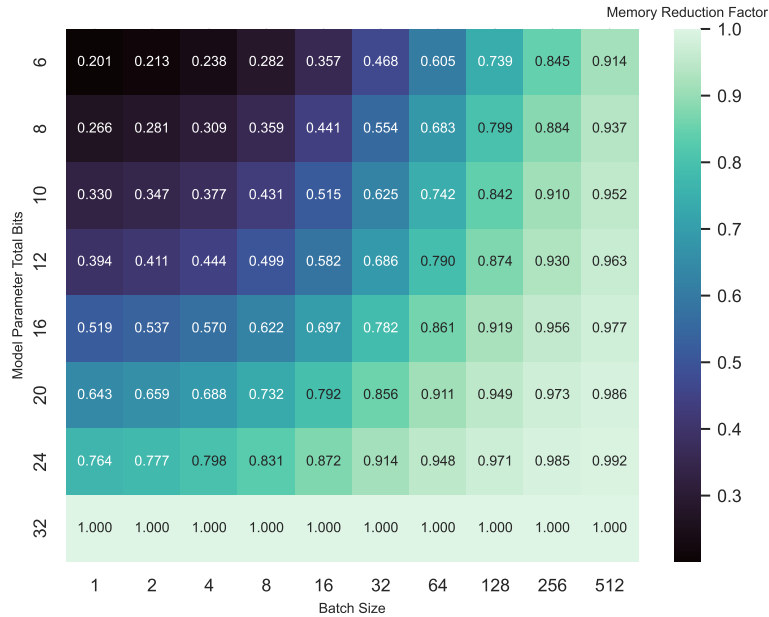


Figure 2: Memory Reduction depending on total bit width and batch size compared to *STE*

Param Update	<i>STE</i>	<i>SQPU</i>	<i>HTEQPU</i>
Inference RM	207252	22666	22666
Param Update RM	0	184586	184586
Sum	207156	207156	207156

Table 3: Number of rounding operations depended on the rounding mode *RM*

presented in Figure 4.

First, we describe Figure 3. Independent of the parameter update method, the *FxP* configurations with 32 and 24 total bits perform comparably to the baseline. For fewer than 16 total bits, the *HTEQPU* methods drop to under 12% for all runs. The best validation accuracy of *SQPU* always equals that of *STE*. Neither approach matches the baseline’s validation accuracy at 8 total bits. At 12 total bits and 6 fractional bits, performance reaches the baseline level, but increasing the number of fractional bits to 8 reduces the accuracy to 65%.

Figure 4 presents the worst and best validation accuracies for different *FxP* configurations, categorized by the parameter update method when using *SR* for inference. We compare this configuration because *STE* and *SR* entail the same number of stochastic rounding operations. Notably, *HTEQPU* slightly outperforms other methods in terms of best validation accuracy when the total bit width is 16 or less, but never exceeds an 18% margin. For other accuracies, *HTEQPU* closely resembles the inference results

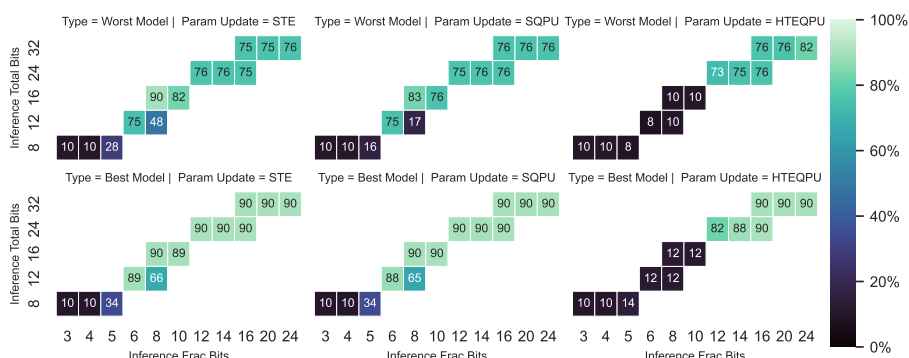


Figure 3: Worst & Best Model Validation Accuracy using HTE in inference for separated by the parameter update method

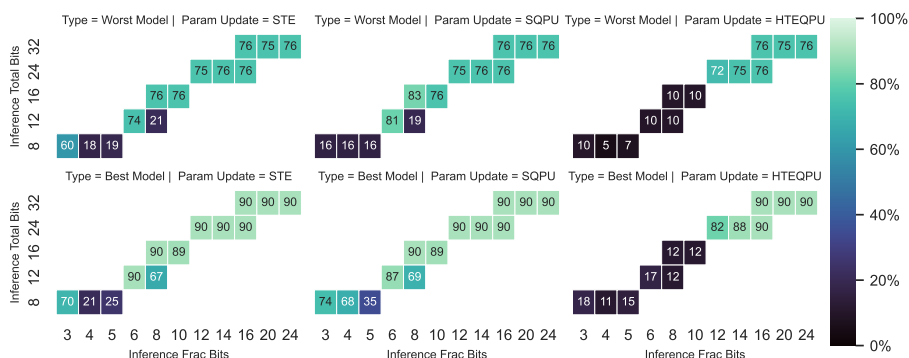


Figure 4: Worst & Best Model Validation Accuracy using SR in inference for separated by the parameter update method

under HTE. SQPU shows almost equal performance for configurations with 16 or more total bits. STE also achieves identical performance at 24 and 32 total bits, but its worst validation accuracy drops to 76% at 16 total bits.

For 12 total bits and 8 fractional bits, SQPU increases the best-model accuracy by 7%, reaching 81%. At 8 total bits, both STE and SQPU achieve their highest accuracies with 3 fractional bits, reaching 74%, though SQPU’s worst-model accuracy is just 16%. With 4 fractional bits, SQPU attains a best-model accuracy of 68%, while STE reaches only 21%. At 5 fractional bits, STE underperforms relative to HTE during inference, whereas SQPU maintains comparable performance. Although SQPU can often match or surpass STE, the gap between its worst and best accuracies increases considerably. This suggests that training with STE is more stable than SQPU at lower bit widths.

We conclude our findings from the first experiment as follows. All approaches match the full-precision baseline’s accuracy for certain FxP configurations. Among QPU variants, SQPU generally outperforms HTEQPU. At lower bit widths, SQPU can sometimes surpass and sometimes trail STE. We also observe that using SR for inference yields better performance at lower bit widths; thus, we focus on SR in subsequent

experiments.

We suspect that applying one quantization scheme to both parameters and layer outputs may be suboptimal because accumulation can lead to larger values. This inference is reinforced by observations indicating that lower bit widths require careful selection of the fractional length. Furthermore, we find that the full range of possible parameter values is often underutilized, suggesting that the approach from [Gupta et al. 2015], which proposes using higher precision for weights than for network outputs, may yield improved results.

For total bit widths above 12, the standard deviation is typically 6% across all approaches. At lower bit widths, *HTEQPU* does not achieve comparable performance, and *SQPU* exhibits greater stability issues than *STE*.

Because of the observed instabilities, we examined each individual training run. We found that both *STE* and *SQPU* can experience training collapse; however, *SQPU* tends to collapse earlier in the training phase, while *STE* usually collapses slightly later. We believe that certain parameters become excessively large, causing layer outputs to overflow. We hypothesize that if this occurs repeatedly in the network, training may collapse because some gradients explode while others vanish. Furthermore, we suspect that using a static learning rate, which is no longer state-of-the-art, contributes to these instabilities.

Training on the edge invariably poses a challenge because unstable neural network training is problematic. Consequently, our next goal is to stabilize the training. Since our experiments thus far have been conducted only with a batch size of 256, we will now provide results for different batch sizes. This is especially relevant since our proposed approach offers the greatest memory reduction at lower bit widths and smaller batch sizes.

## 4.2 Increase Stability of Training

Based on the findings from our initial experiment, we have slightly revised our experimental setup. Specifically, we decrease the fractional length of each layer’s output to increase its range, thereby reducing the likelihood of overflow. We anticipate that this approach partially explains why 8 total bits perform better with fewer fractional bits. However, the parameter range remains limited. To address this, we add a parameter decay of 0.0005 as an L2 penalty, further mitigating potential overflows. In addition, we introduce a learning-rate scheduler that halves the learning rate whenever the validation loss does not improve for five consecutive epochs.

Next, to verify training performance across different batch sizes, we run each experiment with batch sizes of 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, and 1, demonstrating how much memory consumption can be reduced while still achieving comparable performance. Because reducing the learning rate may require more training steps, we increase the number of epochs to 100. We also introduce early stopping after 10 epochs of no improvement in validation loss or accuracy, while ensuring each model is trained for a minimum of 20 epochs. We acknowledge that five seeds are insufficient for statistical significance, so we increase the seed count to 10, striking a balance between robustness and computational cost. We also retrain the full-precision baseline model 10 times for every batch size. In total, this yields 2,750 different training runs.

To reduce the computational cost of the experiment, we limit the number of *FxP* configurations to eight (see Table 4). Based on our positive results, we also include a 6-bit total configuration. Furthermore, we always apply stochastic rounding in the inference phase because it outperforms *HTE* at lower bit widths, yet we retain *HTE* as a quantized

Parameter / Inference Total Bits	32	24	20	16	12	10	8	6
Parameter Fractional Bits	20	16	12	10	8	6	5	4
Inference Fractional Bits	16	12	10	8	6	4	3	2

Table 4: Reduced set of Bit Widths for the experiment

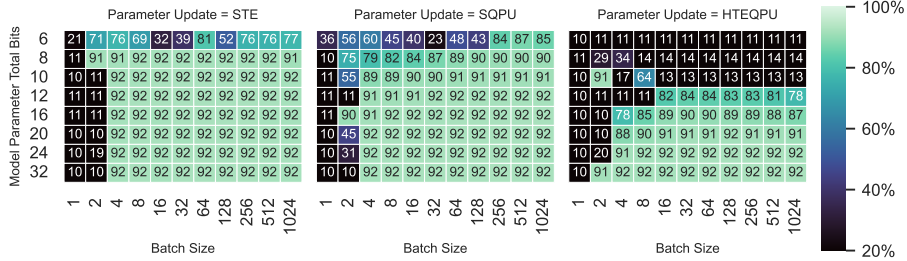


Figure 5: Maximum Validation Accuracy using SR in inference for separated by the parameter update method

parameter update method. Our memory consumption remains unchanged apart from storing the last five validation losses for the learning-rate scheduler. Because these five additional values are insignificant compared to the overall number of stored parameters, we exclude them from further discussion. The full-precision baseline model’s maximum accuracy improves from 90% at a batch size of 4 to 92% at batch sizes greater than 128. For batch sizes of 1 and 2, the baseline’s maximum accuracy is only 10%.

In Figure 5, we illustrate the highest validation accuracy achieved over 10 runs. Each graph displays the same *FxP* configuration under different parameter update methods. The x-axis represents the batch size, while the y-axis denotes the number of total bits. The corresponding fractional bits for parameters and inference are listed in Table 3. The number in each heat map cell indicates the validation accuracy, and the color mapping provides a more convenient overview.

Generally, each new training run surpasses our previous results. The *STE* method achieves approximately 92% validation accuracy across all quantization schemes, provided the batch size is not below 4. If the batch size is less than 4, the accuracy approaches 10%. The *SQPU* approach attains comparable performance for higher total bit widths and batch sizes greater than 2. However, for total bit widths below 16, the accuracy degrades more noticeably as batch sizes get smaller. For more than 10 total bits and a batch size above 2, the accuracy remains at least 91%. At 10 total bits and batch size 4, we observe a 3% drop in accuracy; at 8 total bits and batch size 64, the best-model accuracy dips by 4%. For 6 total bits, training performance deteriorates considerably at batch sizes under 256. Likewise, *HTEQPU* also matches baseline performance when total bits exceed 16 and the batch size is above 6. Here, both smaller batch sizes and lower total bits increase the risk of model degradation.

To assess the stability of the training runs, we present the standard deviation of the validation accuracy across 10 training runs in Figure 7. We consider the standard deviation sufficiently indicative of training stability; however, for completeness, we also provide the mean validation accuracy in Figure 6. The standard deviation varies primarily at the edge of the Pareto front.

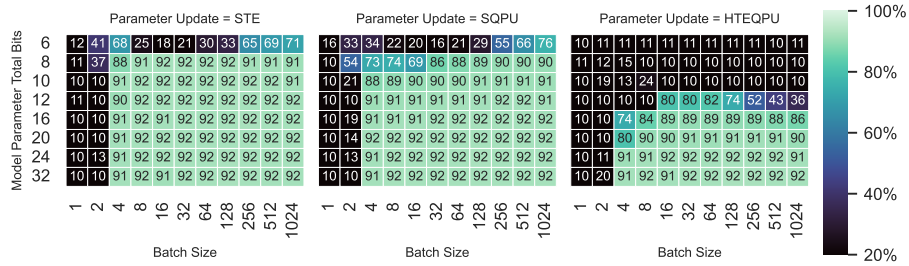


Figure 6: Mean Validation Accuracy using SR in inference for separated by the parameter update method

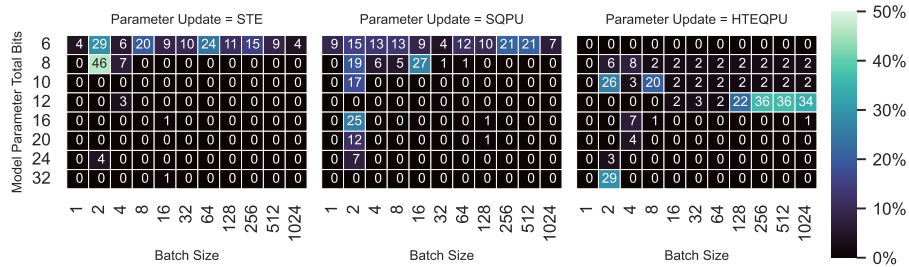


Figure 7: Standard Deviation of the Validation Accuracy using SR in inference for separated by the parameter update method

The main findings from this experiment are as follows. The stability of the training process has markedly improved, with the standard deviation in most runs converging to nearly 0%. Notably, higher standard deviations occur in training runs that approach the threshold of performance degradation, indicating that such runs are more prone to instability. Compared to our previous experiment, the maximum validation accuracy (independent of the parameter update method) increases from 90% to 92%. For all methods (*STE*, *SQPU*, and *HTEQPU*), validation accuracy decreases when the batch size is below 8 or the total bit width is below 6. The bit width exerts a more gradual effect on model degradation, whereas the batch size has a more pronounced impact on accuracy. Although training with 6 total bits is feasible, it incurs reduced validation accuracy. The highest accuracy for 6 total bits occurs with *SQPU*, for batch sizes exceeding 128, showing a moderate degradation of 5–8%. With batch sizes below 4, training outcomes become highly variable or even fail entirely. We attribute this to averaging in the loss function, which amplifies the effect of individual samples within small batches. Consequently, an unfavorable sample sequence can eliminate virtually all training progress. We believe a more robust learning-rate initialization would substantially improve performance and stability for all training runs, even under challenging conditions.

### 4.3 Batch size dependent learning rate initialization

Much research has focused on optimizing the learning rate in relation to the batch size, as shown by studies such as [Goyal et al. 2017, Krizhevsky 2014, Jastrzębski et al. 2018].

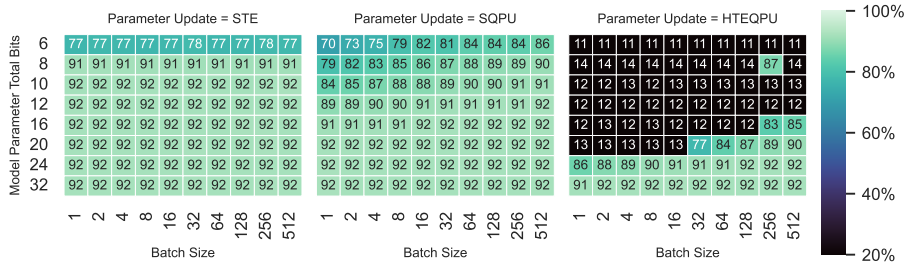


Figure 8: Maximum Validation Accuracy using SR in inference for separated by the parameter update method with Learning Rate scaling in respect to the batch size

We adopted the method proposed by [Goyal et al. 2017], which scales the learning rate linearly with the batch size. We chose this approach for its simplicity and minimal overhead, as it does not incur additional memory costs. By excluding the experiment with batch size 1024—where learning-rate scaling has no effect—we comprehensively evaluated the method’s effectiveness across 2,500 experiments, providing a thorough assessment of its performance.

Our full-precision baseline model achieved beyond 91%, without further improvement in maximum accuracy. In Figure 8, we show the highest accuracy obtained across 10 seeds; the figure follows the same format as Figure 5.

STE now produces strong results for all FxP configurations exceeding 6 bits, irrespective of the batch size. However, at 6 bits, the validation accuracy decreases to approximately 78%. SQPU continues to display similar or superior accuracies across all FxP configurations compared to previous results. Nonetheless, smaller batch sizes and fewer total bits still reduce performance. For 6 bits, SQPU outperforms STE by up to 9% at larger batch sizes, but is outperformed at batch sizes below 8.

For HTEQPU, we see comparable results at 32 bits across all batch sizes. At 24 bits, smaller batch sizes degrade model performance. At 20 bits, acceptable performance emerges only at large batch sizes. For fewer than 16 total bits, training halts entirely, faring worse than our previous observations. Interestingly, one training run with 8 total bits at a batch size of 256 reaches 87% validation accuracy, suggesting that successful training could be feasible. Combined with the poorer results at 20 and 16 bits, this points to the possibility that HTEQPU training might be improved by tuning different hyperparameters.

As in the previous section, we present both the mean validation accuracy and the standard deviation across the 10 training runs. Overall, batch size appears to have minimal influence on training stability. For STE, the standard deviation at 6 total bits can reach up to 4%. In contrast, SQPU exhibits higher variability at the same bit width, with standard deviations ranging from 2% to 19%. This suggests that further hyperparameter tuning may improve the consistency of SQPU.

We also observe that most of the top-performing runs surpass our previous results. STE achieves 90% accuracy almost everywhere, which also holds for most SQPU runs and some HTEQPU runs. In fact, STE consistently exceeds 91% in its best runs. Smaller batch sizes generally degrade performance. Previously, SQPU showed more pronounced stability issues at lower bit widths; now, these problems appear primarily for small batch sizes.

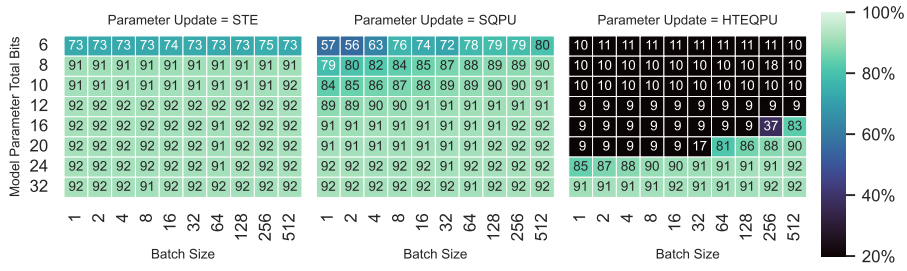


Figure 9: Mean Validation Accuracy using SR in inference for separated by the parameter update method with Learning Rate scaling in respect to the batch size

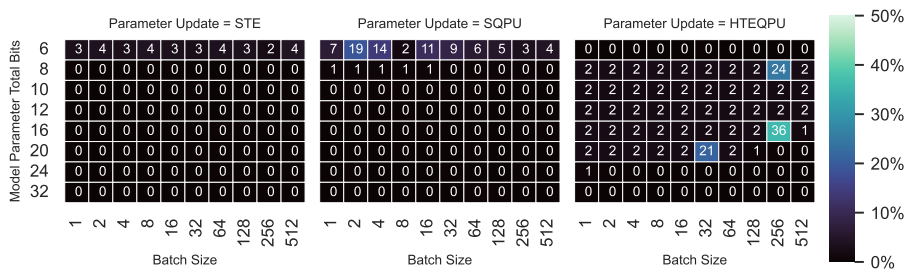


Figure 10: Standard Deviation of the Validation Accuracy using SR in inference for separated by the parameter update method with Learning Rate scaling in respect to the batch size

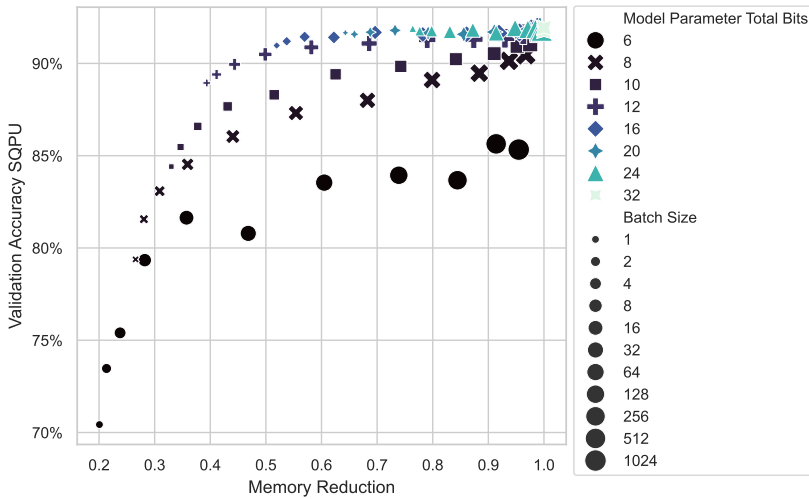


Figure 11: Maximum Validation Accuracy of SQPU in respect to the memory reduction factor

Finally, we show the maximum validation accuracy of our *SQPU* and *STE* approaches in relation to the estimated memory reduction during training (see Figure 11). Lower memory reductions favor our approach compared to *STE*. Specifically, *SQPU* attains near-baseline accuracy while consuming only 0.57% of the training memory at 16 total bits for the model parameters and a batch size of 16. Furthermore, we can reduce the memory to below 50% with a relatively small accuracy drop of 1.5% using 12 total bits for the model parameters and a batch size of 16.

## 5 Conclusion

In this paper, we introduced a novel approach called Quantized Parameter Updates (*QPU*) as an alternative to the Straight-Through Estimator (*STE*) for training models with reduced memory footprints. The key innovation lies in using stochastic rounding for quantized parameter updates (*SQPU*), which yields more consistent and improved training outcomes. Our method, employing higher precision for parameters than for layer outputs, effectively limits overflow risks. We also enhanced training stability by incorporating L2 regularization in the form of weight decay and an adaptive learning rate scheduler that reacts to training plateaus. In addition, reducing the initial learning rate relative to the batch size boosts validation accuracy, especially for smaller batch sizes and fixed-point (*FxP*) configurations with fewer total bits.

We validated our approach on the FASHION-MNIST dataset, showing that it achieves accuracy comparable to *STE* while cutting memory usage to just 57% during training. By allowing a tolerable 1.5% accuracy drop, we reduce memory usage to 50%. This work enables on-the-edge training and remains straightforward enough to be used without much fine-tuning of quantization schemes, making it potentially suitable for high-performance embedded systems. Nevertheless, gradient-related computations are still expensive. In our future research, we plan to integrate quantized gradients into the approach, further lowering computational costs and broadening applicability across diverse models and datasets. Additionally, we envision that stochastic quantized parameter updates (*SQPU*) could be employed beyond fixed-point (*FxP*) quantization schemes.

## References

- [Bengio et al. 2013] Bengio, Y., Léonard, N. & Courville, A. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv*. (2013)
- [Bottou et al. 1994] Bottou, L., Cortes, C., Denker, J., Drucker, H., Guyon, I., Jackel, L., LeCun, Y., Muller, U., Sackinger, E., Simard, P. & Vapnik, V. Comparison of classifier methods: a case study in handwritten digit recognition. *Proceedings Of The 12th IAPR International Conference On Pattern Recognition*. (1994)
- [Fox et al. 2021] Fox, S., Rasoulinezhad, S., Faraone, J., Boland & Leong, P. A Block Mini-float Representation for Training Deep Neural Networks. *International Conference On Learning Representations*. (2021), <https://openreview.net/forum?id=6zaTwpNSsQ2>
- [Gennari do Nascimento et al. 2023] Gennari do Nascimento, M., Adrian Prisacariu, V., Fawcett, R. & Langhammer, M. HyperBlock Floating Point: Generalised Quantization Scheme for Gradient and Inference Computation. *2023 IEEE/CVF Winter Conference On Applications Of Computer Vision (WACV)*. (2023)
- [Goyal et al. 2017] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. & He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv*. (2017)

- [Grahn et al. 2014] Grahn, P., Mallory, G., Berry, M., Hachmann, J., Lobel, D. & Lujan, L. Restoration of motor function following spinal cord injury via optimal control of intraspinal microstimulation: toward a next generation closed-loop neural prosthesis. *Frontiers In Neuroscience*. (2014)
- [Guo et al. 2019] Guo, J., Liu, W., Wang, W., Yao, C., Han, J., Li, R., Lu, Y. & Hu, S. AccUDNN: A GPU Memory Efficient Accelerator for Training Ultra-Deep Neural Networks. *2019 IEEE 37th International Conference On Computer Design (ICCD)*. (2019)
- [Gupta et al. 2015] Gupta, S., Agrawal, A., Gopalakrishnan, K. & Narayanan, P. Deep learning with limited numerical precision. *Proceedings Of The 32nd International Conference On International Conference On Machine Learning - Volume 37*. (2015)
- [Im and Kim 2020] Im, M. & Kim, S. Neurophysiological and medical considerations for better performing microelectronic retinal prosthesis. *Journal Of Neural Engineering*. (2020)
- [Jacob et al. 2018] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. & Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *2018 IEEE/CVF Conference On Computer Vision And Pattern Recognition*. pp. 2704-2713 (2018)
- [Jastrzębski et al. 2018] Jastrzębski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y. & Storkey, A. Width of Minima Reached by Stochastic Gradient Descent is Influenced by Learning Rate to Batch Size Ratio. *Artificial Neural Networks And Machine Learning – ICANN 2018*. pp. 392-402 (2018)
- [Kathe et al. 2022] Kathe, C., Skinnider, M., Hutson, T. & Al. The neurons that restore walking after paralysis. *Nature*. (2022)
- [Krizhevsky 2009] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. (2009), <https://www.cs.toronto.edu/~kriz/cifar.html>
- [Krizhevsky 2014] Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv*. (2014)
- [Lecun et al. 1998] Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings Of The IEEE*. 86, 2278-2324 (1998)
- [Li et al. 2014] Li, Y., Shi, D., Ding, B. & Liu, D. Unsupervised Feature Learning for Human Activity Recognition Using Smartphone Sensors. *Lecture Notes In Computer Science*. (2014)
- [Micikevicius et al. 2018] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G. & Wu, H. Mixed Precision Training. *International Conference On Learning Representations*. (2018), <https://openreview.net/forum?id=r1gs9JgRZ>
- [Netzer et al. 2011] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B. & Ng, A. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop On Deep Learning And Unsupervised Feature Learning 2011*. (2011), <http://ufldl.stanford.edu/housenumbers/nips2011>
- [Rastegari et al. 2016] Rastegari, M., Ordonez, V., Redmon, J. & Farhadi, A. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *Lecture Notes In Computer Science*. pp. 525-542 (2016)
- [Russakovsky et al. 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. & Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal Of Computer Vision (IJCV)*. (2015)
- [Sun et al. 2018] Sun, J., Fu, Y., Li, S., He, J., Xu, C. & Tan, L. Sequential Human Activity Recognition Based on Deep Convolutional Network and Extreme Learning Machine Using Wearable Sensors. *Journal Of Sensors*. (2018)
- [Sun et al. 2019] Sun, X., Choi, J., Chen, C., Wang, N., Venkataramani, S., Srinivasan, V., Cui, X., Zhang, W. & Gopalakrishnan, K. Hybrid 8-bit floating point (HFP8) training and inference for

deep neural networks. *Proceedings Of The 33rd International Conference On Neural Information Processing Systems*. (2019), <https://dl.acm.org/doi/10.5555/3454287.3454728>

[Xiao et al. 2017] Xiao, H., Rasul, K. & Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv*. (2017), <https://github.com/zalandoresearch/fashion-mnist>

[Yang et al. 2019] Yang, G., Zhang, T., Kirichenko, P., Bai, J., Wilson, A. & De Sa, C. SWALP : Stochastic Weight Averaging in Low-Precision Training. *arXiv*. (2019)

[Zhou et al. 2016] Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H. & Zou, Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv*. (2016)