TOWARD EFFICIENT LOW-PRECISION TRAINING: DATA FORMAT OPTIMIZATION AND HYSTERESIS QUANTIZATION

Anonymous authors
Paper under double-blind review

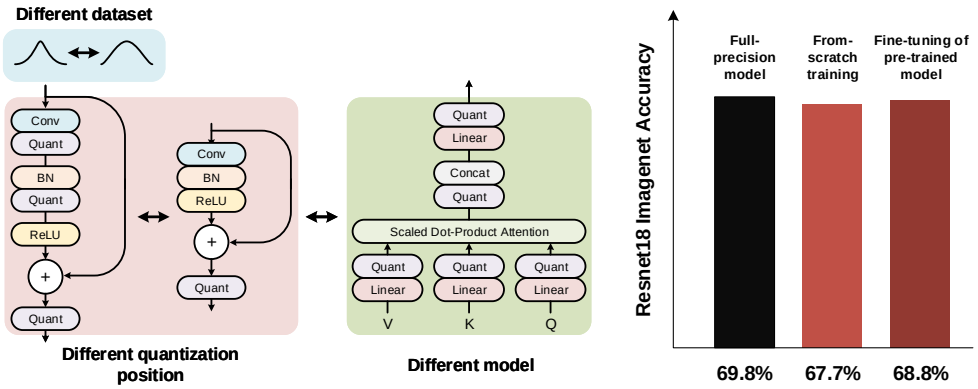
ABSTRACT

As the complexity and size of deep neural networks continue to increase, low-precision training has been extensively studied in the last few years to reduce hardware overhead. Training performance is largely affected by the numeric formats representing different values in low-precision training, but finding an optimal format typically requires numerous training runs, which is a very time-consuming process. In this paper, we propose a method to efficiently find an optimal format without actual training of deep neural networks. We employ this method to determine an 8-bit format suitable for the training of various models. In addition, we propose a new quantization scheme that utilizes the hysteresis effect to suppress undesired weight fluctuation during training. This quantization scheme enables deeply quantized training using 4-bit weights, exhibiting only 0.2% degradation for ResNet-18 trained on ImageNet.

1 INTRODUCTION

Deep neural networks have been used in various fields such as vision, audio, natural language processing, and reinforcement learning. As larger and more complex neural networks are adopted, the energy and time consumed for training have become a critical issue in hardware implementation. Using low-bit representations in training significantly reduces hardware overhead and memory footprint; hence, neural network training with limited precision has been extensively studied recently. For instance, 16-bit formats are already adopted in commercial devices such as FP16 (IEEE, 2019) in NVidia GPU and bfloat16 (Kalamkar et al., 2019) in Google TPU (Wang et al., 2019). Also, it has been demonstrated that even 8-bit formats could be adopted in deep neural network training with reasonable accuracy (Sun et al., 2019; Fox et al., 2020). However, there are various issues in realizing low-precision training in practical applications as detailed below.

Optimal data format for low-precision training: Training performance is susceptible to the data format we use to represent variables in the network. When a value is represented using a fixed number of bits, there is a trade-off between dynamic range and precision. For instance, allocating more bits to the exponent part in a floating-point format enlarges the dynamic range but lowers precision due to fewer bits in the mantissa part. Recent studies on 8-bit training suggest various ways to reduce the dynamic range required for number representation to enhance representation precision. Early work on 8-bit training (Wang et al., 2018) adopts a 5-bit exponent to represent different variables using a single format, but Sun et al. (2019) examine the statistics of each variable and optimize the numeric formats separately. Specifically, the values used in the forward path (weight and activation) have a relatively narrow dynamic range, and only 4 bits are allocated to the exponent. Fox et al. (2020) propose to divide data into smaller blocks and assign a shared exponent bias to each block. Since the values in a block tend to exhibit similar statistics, the forward (weight and activation) and backward (error) paths could be represented using only 2-bit and 4-bit exponents, respectively. However, these approaches are difficult to generalize since we should empirically decide numeric formats for each task, neural network structure, and quantization scheme (Fig. 1). Furthermore, analyzing the statistics of each variable is not enough to determine an optimal format. Their distributions often have a long tail, and hence the dynamic range of the numeric format should be experimentally selected through many trial-and-errors in actual training.



(a) Various quantized training environments

(b) Performance degradation in quantized models

Figure 1: Two Challenges in low-precision training. (a) An optimal format varies with dataset, quantization scheme, and model. (b) There is a performance gap between from-scratch training with quantized weights and fine-tuning of a pre-trained full-precision model.

Performance degradation in from-scratch training: Low-precision training differs from obtaining quantized neural network models. A quantized neural network refers to the model with quantized parameters (e.g., weight). The target model is first trained in full precision, and then this pre-trained model is further fine-tuned while the weights are now kept quantized, minimizing performance degradation by making the network adapt to weight quantization. Previous studies show that a model could achieve comparable accuracy to full-precision models even using 1- or 2-bit weights (Choi et al., 2019; Martinez et al., 2020) through fine-tuning a pre-trained model. Contrarily, in low-precision training, the entire process of neural network training must be processed in a deeply quantized environment; the neural network is trained from scratch while all the values in the network - not only parameters, but also other variables in the network (e.g., activation, error, and weight gradient) - are deeply quantized. The quantized model from low-precision training typically shows a noticeable accuracy drop compared to the quantized model obtained from fine-tuning of a pre-trained model (Elhoushi et al., 2021). Fig. 1(b) shows the Top-1 validation accuracy of ResNet-18 (He et al., 2016) trained on ImageNet (Deng et al., 2009) for different training schemes. The weights are quantized into a 4-bit base-2 logarithmic format. From-scratch training of the model with quantized weights results in a 2.1% accuracy drop, whereas only 1.0% degradation is observed if we fine-tune a pre-trained model. This suggests that even though a better solution exists for a given format, it cannot be reached through from-scratch training.

To formalize the issues above, here we divide quantization in low-precision training into two types: network quantization and data flow quantization. Network quantization refers to the quantization of the neural network model. An example of this type of quantization is weight quantization. In network quantization, we need to reduce the performance difference between from-scratch training and fine-tuning (Yang et al., 2019b). On the other hand, data flow quantization refers to the on-the-fly quantization that occurs when data propagate through the network in low-precision training. Examples include activation, error, and weight gradient quantizations. Additional errors are introduced in weight update computation due to this type of quantization, which leads to performance degradation. Hence, we need to find an optimal format to minimize accuracy drop due to computation errors in data flow quantization.

In this paper, we present a systematic approach to implementing low-precision training on various models and tasks. First, we present a method to efficiently find an optimal format for data flow quantization. In addition, we introduce a hysteresis quantization technique, a new quantization method for network quantization that can mitigate the issues of from-scratch training. Our main contributions are:

- We present **a method that can predict the training performance of various numeric formats for data flow quantization**. This method allows us to determine an appropriate data format for different neural network structures, datasets, and tasks efficiently.
- Using the method above, we propose **an optimal 8-bit format** suitable for low-precision training of various models, which enables quantization of BatchNorm layer input and improves hardware efficiency with minimal performance degradation.
- We propose **a new quantization scheme that utilizes the hysteresis effect** to improve the performance of from-scratch training. This scheme enables ultra-low-precision training using 4-bit logarithmic weights.

2 Data Flow Quantization

2.1 Numeric Formats

There are many numeric formats that can be constructed with *n* bits depending on how much dynamic range is required and how many valid bits are used for representing a value. For example, using 8 bits we could implement 8-bit fixed point integer format, 8-bit floating-point formats such as FP152, FP143, and FP125 (FP1*xy* represents 1 sign bit, *x* exponent bits, *y* mantissa bits), 8-bit posit format (Gustafson & Yonemoto, 2017), and 8-bit float-fix format (Han et al., 2019). Since the diversity of formats that could be formulated using *n* bits is nearly unlimited, here we assume some constraints to limit candidates while still including widely used formats such as fixed-point and floating-point formats as below:

- The MSB (Most Significant Bit) is used as a sign bit and other bits represent magnitude. Accordingly, only symmetric formats are considered. Two's complement representation is slightly asymmetric, but it does not incur a significant difference.
- The number of valid bits of a larger value is greater than or equal to the number of valid bits of a smaller value. The valid bits stand for significant digits in binary representation.
- The ratio between consecutive representable values does not exceed 2. For example, the base-4 logarithmic format is excluded.

We could obtain 166 8-bit formats that meet these constraints. Then, we reduce 1 and 2 valid bits in each format to obtain 7- and 6-bit formats, resulting in 498 formats in total. More information on the numeric formats considered in our experiments is provided in Appendix A.1.

2.2 Activation and Error Quantization

In a neural network consisting of *n* layers, the training process is described by

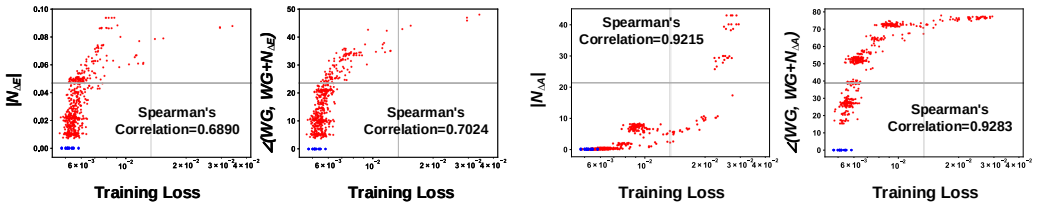
$$A_{l+1} = f_l(W_l^t, A_l) \tag{1}$$

$$E_l = g_l(W_l^t, E_{l+1}) \tag{2}$$

$$WG_l = h_l(A_l, E_{l+1}) \tag{3}$$

$$W_l^{t+1} = o(WG_l, W_l^t) \tag{4}$$

where A, E, W, and WG are activation, error, weight, and weight gradient, respectively. f, g, h, and o are forward, backward, gradient, and update functions. l and t represent the layer number and time step. We follow the quantized training scheme suggested by Fox et al. (2020), but with the following modifications to reduce hardware implementation costs. A and E are quantized not only for the GEMM input but also for the BatchNorm layer input. BatchNorm layer normalizes input using the mean and variance of each channel, but these values are obtained only after observing all the inputs from the previous layer, necessitating that all input values are temporarily stored in memory. Therefore, quantizing the BatchNorm layer's input significantly reduces memory footprint and memory access overhead. Additionally, the scope of sharing exponent bias is extended to a layer (A_l and E_l) to avoid the overhead of aligning partial sums from different blocks in block-wise exponent sharing. Finally, instead of determining the shared exponent bias by analyzing all values in the layer, we conservatively update it by detecting overflow and underutilization that occurred in the previous mini-batch.



(a) Error quantization

(b) Activation quantization

Figure 2: Training loss vs. proposed performance indicators. Blue dots represent full-precision training runs and red dots represent training runs with 8-, 7-, and 6-bit quantization formats.

2.3 Indicators of Training Performance

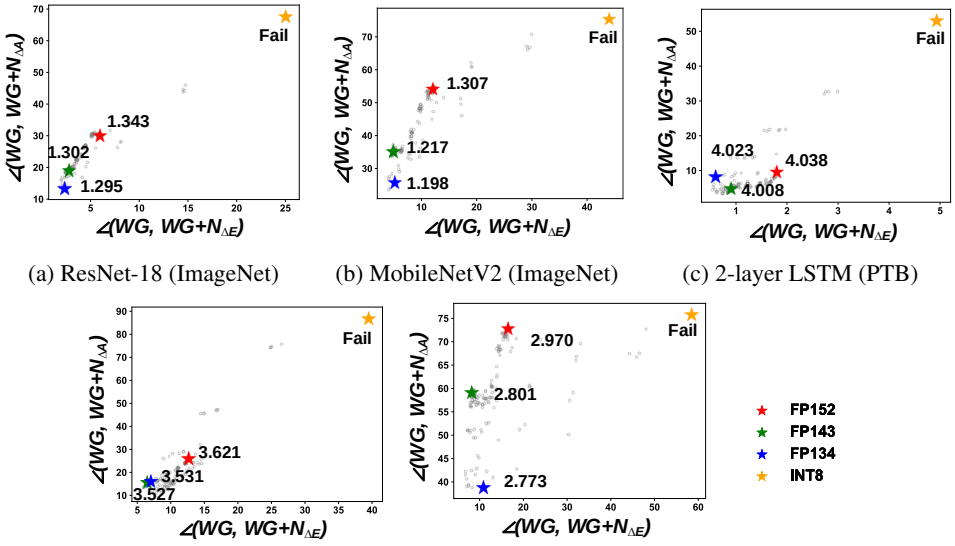
Effect of quantized error: Quantizing the error E in the backward path is independent of how the forward path behaves since the loss surface of the model does not change. Therefore, the optimal W that the network needs to reach through training remains the same regardless of the error quantization scheme. However, when the error is quantized, a quantization error ΔE is introduced in E, which incurs a noise $N_{\Delta E}$ in WG through the gradient function in Eq. 3 and potentially updates each weight in the wrong direction. Therefore, we suggest that the weight gradient error $N_{\Delta E}$ could be a good indicator of degradation in training performance. One way to implement this is predicting performance using the magnitude of $N_{\Delta E}$; however, if the noise is in the same direction as WG, it would only change the amount of each update and result in a less severe effect. Instead, we propose to measure the misalignment between $WG + N_{\Delta E}$ and WG for performance prediction. The misalignment between two vectors is estimated by

$$\angle(A, B) = cos^{-1}\left\{\frac{A \cdot B}{\|A\|_2 \cdot \|B\|_2}\right\} \tag{5}$$

Then, the change in the update direction due to $N_{\Delta E}$ is $\angle(WG, WG + N_{\Delta E})$. We can expect that the smaller $\angle(WG, WG + N_{\Delta E})$, the better the training performance.

Effect of quantized activation: Contrary to error quantization, activation quantization affects the way the forward path operates, and the loss surface of the model changes. Hence, the global optima of weight parameters shift, where the amount of shift would be proportional to the quantization noise. The displacement of global optima can be indirectly estimated using the direction of the weight gradients WG. If the angle $\angle(WG, WG + N_{\Delta A})$ is small, the deviation of the global optima is expected to be small as well, suggesting a better training performance.

In the discussions above, we assumed that the angles $\angle(WG, WG + N_{\Delta E})$ and $\angle(WG, WG + N_{\Delta A})$ could be used to predict training performance. We experimentally prove this by comparing the training performance of different numeric formats. For 498 numeric formats in 6 to 8 bits, we compare the loss obtained from training with the proposed performance indicators. The results are summarized in Fig. 2. Training loss is obtained by training ResNet-18 on CIFAR-10 dataset using SGD with a momentum of 0.9 for 60 epochs. The batch size is 128 images and the initial learning rate is 0.1, which is decayed by a cosine scheduler. Then, we measure each angle by processing 100 mini-batches on this pre-trained model. Note that we use WG of the first layer's weight since it reflects the activation and quantization errors of all the layers in the network. Fig. 2 shows that the training performance has a higher Spearman's rank correlation with the misalignment angle than the magnitude of the error in the weight gradient, suggesting that the misalignment angle is a better metric to predict the ranking of various formats based on the training performance. Using this method, we could determine the optimal format for a specific neural network model, dataset, and task very efficiently as we only need to measure the misalignment angle between the two weight gradients without time-consuming network training. For experiments in Fig. 2, the amount of computation is reduced by 99.6%, and the reduction will be even larger for larger datasets and complex networks that need more epochs for training.



(a) ResNet-18 (ImageNet) (b) MobileNetV2 (ImageNet) (c) 2-layer LSTM (PTB)

(d) Transformer (IWLST) (e) MobileNetV2 + SSDLite (VOC)

Figure 3: Scatter plots displaying misalignment angles for 166 8-bit formats. The numbers next to four selected data formats (INT8, FP152, FP143, and FP134) represent the loss measured in actual training.

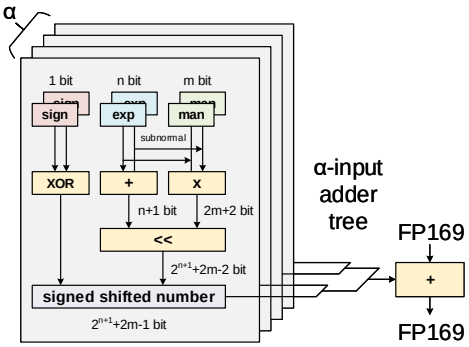


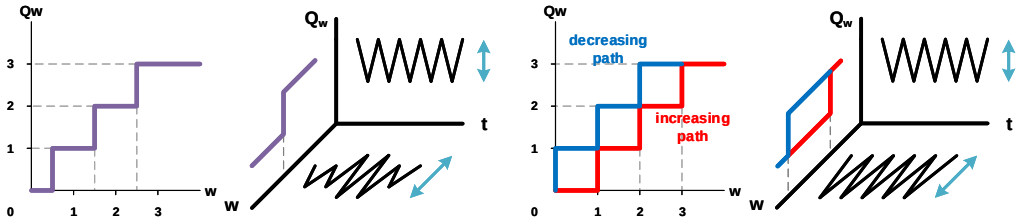
Figure 4: Multi-way MAC unit with an adder tree.

Table 1: Area of 2-stage pipelined MAC units synthesized in a 40nm process.

	Area per MAC [μm^2]		
	FP134	FP143	FP152
Conventional MAC	1355	1320	1310
Multi-way MAC			
1-input	2285	2566	3185
2-input	1335	1480	2212
4-input	888	1034	1571
8-input	678	836	1197
16-input	571	698	989
32-input	511	668	909
64-input	509	638	854

2.4 Optimal Format for Data Flow Quantization

Using the proposed performance estimation method above, here we find an optimal format suitable for training various models. We select five models widely used in quantized training research for experiments: ResNet-18, MobileNetV2 (Sandler et al., 2018), 2-layer LSTM, small transformer for translation on the IWSLT German to English dataset (Cettolo et al., 2014), and SSD-Lite (Liu et al., 2016) with MobileNetV2. We first measure misalignment angles ($\angle(WG, WG + N_{\Delta E})$ and $\angle(WG, WG + N_{\Delta A})$) for 166 8-bit formats. To verify the correlation between the training performance and the misalignment angles, we select four formats that exhibit low hardware implementation costs (INT8, FP152, FP143, and FP134) and train the networks using each format. While we may use different formats for activation and error, it requires a complicated datapath (Sun et al., 2019) and hence we only consider a single format for both variables. The experimental results in Fig. 3 demonstrate that the training performance is higher if both misalignment angles are small in all tasks and models, confirming that the proposed indicators could be used to determine the optimal numeric format.



(a) Conventional quantization

(b) Hysteresis quantization

Figure 5: Comparison of quantization schemes.

From the results in Fig. 3, we could conclude that FP134 and FP143 are the best options across all models. Although they both use 8 bits, their hardware implementation costs may differ significantly. For comparisons, we implemented a MAC unit using those formats as depicted in Fig. 4. For accumulation, we use FP169 which was confirmed to exhibit no degradation through chunk-based accumulation (Wang et al., 2018). In a conventional MAC unit with a single multiplier, the power and area are dominated by the 16-bit accumulation block, undermining the advantages of using a low-precision data format. Instead, we adopt a multi-way MAC unit with an adder tree proposed by Park et al. (2021) to minimize hardware overhead (Fig. 4). The outputs of multipliers are combined through integer addition, and only one floating-point addition is required to convert the combined value to a partial sum in FP169. In this structure, a smaller dynamic range is advantageous since all multiplication results need to be aligned and summed, and shifter overhead increases with the dynamic range. As a result, FP134 format exhibits the lowest cost as shown in Table 1. Therefore, we suggest that the FP134 format is the optimal format for low-precision training as it results in good training performance in the five models and low hardware cost.

3 Network Quantization

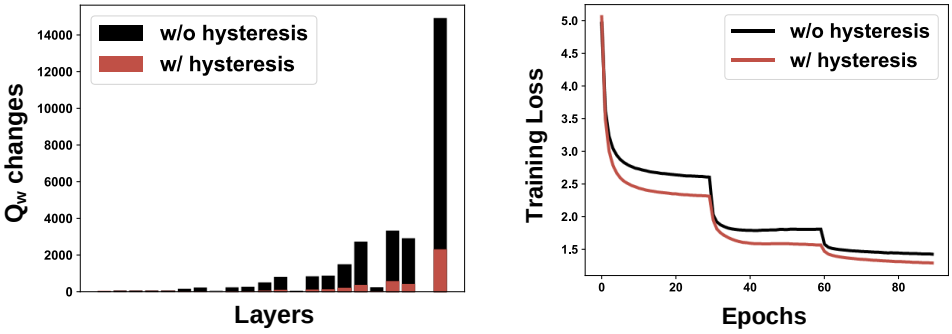
In quantized neural networks, the weight parameters are generally quantized in a way that minimizes the quantization error (Choi et al., 2019; Martinez et al., 2020). For instance, if x is quantized into a fixed-point format through $s\lfloor \frac{x}{s} \rceil$, a proper value is selected for the scaling factor s to minimize the quantization error. However, as the weights continue to change during training, we need to calculate s for every update, which could cause significant overhead. Therefore, prior studies on low-precision training suggest constraining the scaling factor to the power of 2 in the shared exponent (Köster et al., 2017) or the shared exponent bias (Fox et al., 2020). In this section, we analyze the issues behind weight quantization and propose a new quantization scheme to mitigate those issues.

3.1 Fluctuation of Weight Parameters

In typical low-precision training, a master copy of weight parameters is separately maintained in high precision, and those weights are updated based on the computed weight gradient. This high-precision weight is quantized into a low-precision format and used for the forward path computation during training. Since the scaling factor s is constrained to 2^n, the quantization threshold remains the same unless s is updated due to overflow or underutilization. If the optimal weight is located between two representable values of a data format, the quantized weight would fluctuate alternately between the two representable values in each update (Fig. 5(a)). Note that the amount of change in the quantized weight due to this fluctuation is not necessarily proportional to the weight gradient. If the master weight is located close to the quantization threshold, even a very small weight update may cause large fluctuations, undermining training performance.

3.2 Hysteresis Quantization

To mitigate the fluctuation issue above, we propose to introduce the concept of hysteresis to quantization. More specifically, we quantize each weight differently depending on the direction of the weight update. Fig. 5(b) describes the proposed hysteresis quantization scheme. The quantization



(a) Average number of Q_w changes for each update

(b) Learning curve

Figure 6: Experimental results for ResNet-18 training on ImageNet 4-bit logarithmic weights. (a) Comparison of quantized weight change frequency and (b) training performance comparison.

threshold differs when the master weight increases and decreases. For instance, we could round down the weight for a positive weight update, and round it up for a negative weight update. The equation below describes the proposed quantization scheme.

$$Q_w^t = \begin{cases} \lfloor w^t \rfloor, & \text{if } w^t > Q_w^{t-1} \\ \lceil w^t \rceil, & \text{if } w^t < Q_w^{t-1} \end{cases} \tag{6}$$

where w is the original value, Q_w is its quantized value, and t is the time step. The proposed hysteresis quantization reduces fluctuation significantly, stabilizing the training process and allowing the network to reach global optima more efficiently. In addition, even if the weight fluctuates, the weight change is now proportional to the weight update since enough amount of weight update should be accumulated to change the quantized weight. This helps the network to learn better while suppressing fluctuations for small WG values.

3.3 Ultra-Low-Precision Format for Network Quantization

To verify the effectiveness of the proposed hysteresis quantization, we select 4-bit logarithmic representation as an ultra-low-precision format for weight parameters. This format has the same dynamic range as INT8 which is widely used for weight quantization, and is more hardware-efficient as multiplication is implemented only using simple shift operations. There have been attempts to use logarithmic weights in quantized neural networks (Lee et al., 2017; Elhoushi et al., 2021), but from-scratch training shows a significant performance degradation. In logarithmic data formats, the interval of quantization points is not uniform, making them more sensitive to fluctuation.

Fig. 6 shows experimental results of ResNet-18 training on ImageNet using 4-bit logarithmic weights. Note that we apply channel-wise quantization to the convolutional layers to compensate for the insufficient expression range and layer-wise quantization to other types of layers. Further details on the experimental setup are provided in Appendix A.4.1. First, we measure how many quantized weights Q_w change when the network performs one weight update using a mini-batch and average them over the first 100 updates in the 60th epoch. The experimental result displayed in Fig. 6(a) clearly shows that using hysteresis significantly reduces weight change frequency and stabilizes the training process. Fig. 6(b) compares the training performance of quantization with and without hysteresis. Hysteresis quantization not only speeds up training but also achieves better results at the end of training.

4 Experimental Results

In this section, we experimentally show that the proposed training performance prediction method and hysteresis quantization scheme enable efficient low-precision training.

Table 2: Performance Summary of FP134 Data Format

Model (Dataset) [Metric]	Baseline (FP32)	FP134
ResNet-18 (ImageNet)	69.8	69.8
MobileNetV2 (ImageNet)	72.2	71.9
2-layer LSTM (PTB) [ppl.]	91.5	92.0
Transformer (IWSLT) [BLEU]	34.8	34.5
MobileNetV2 + SSDLite (VOC) [mAP]	68.3	68.2

Table 3: Comparisons of Data Formats for Low-Precision Training of ResNet-18 on ImageNet

	Formats (Exponent, Mantissa)						Top-1 Accuracy	
Quantization Scheme	w	GEMM Input x	Batch -Norm Input x	dw	dx	Acc.	FP32	Proposed
SWALP (Yang et al., 2019a)	8[1]	8[1]	-	8[1]	8[1]	32[1]	70.3	65.8
S2FP8 (Cambier et al., 2020)	(5,2)/(8,23)	(5,2)	-	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 (Sun et al., 2019)	(4,3)	(4,3)	(6,9)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 (Fox et al., 2020)	(2,5)	(2,5)	31[1]	(6,9)	(4,3)	31[1]	69.7	69.8
FP8-SEB (Park et al., 2021)	(4,3)	(4,3)	(4,3)	(4,3)	(4,3)	(6,23)	69.7	69.0
FP134 (Ours)	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)	(6,23)	69.8	69.8

[1] Fixed Point

4.1 8-BIT LOW-PRECISION TRAINING

In Section 2.4, we found that FP134 is the optimal format for low-precision training using the proposed performance prediction method. We measure the training performance of this format and compare it against other 8-bit data formats from recent studies by applying those formats to the training of various neural network models. More details on the experimental setup are provided in Appendix A.4. The performance of the proposed data format is summarized in Table 2. Overall, 8-bit training using FP134 achieves nearly the same performance as the full-precision training on all models. Even in MobileNetV2, which is known to be sensitive to quantization due to the small number of parameters, only 0.3% degradation occurred. Sun et al. (2019) show that HFP8 also exhibits only 0.2% accuracy degradation in MobileNetV2 (71.81% vs. 71.61%), but they quantize BatchNorm input into 16 bits instead of 8 bits, roughly doubling the memory access and computational complexity. Additionally, since the forward and backward paths employ different data formats, HFP8 is actually implemented using 9-bit MAC units in hardware (Agrawal et al., 2021). Table 3 compares the training performance of various data formats for ResNet-18 training. Our FP134 format exhibits no accuracy drop compared to full-precision training. HFP8 (Sun et al., 2019) and BM8 (Fox et al., 2020) demonstrate similar performance, but they both use higher precision to represent BatchNorm inputs, and different formats are adopted in the forward and backward paths, necessitating complex computation units when implemented in hardware, as decribed above. In addition, BM8 assumes layer-wise sharing of exponent bias, incurring additional overhead in memory access and data alignment. FP8-SEB (Park et al., 2021) addresses this issue by employing layer-wise exponent bias sharing and multi-way MAC units, but it results in a 0.7% accuracy drop for ResNet-18 training. Contrarily, our data format shows no performance degradation, while deeply quantizing BatchNorm inputs into the same format and allowing for a simple datapath by using an identical data format in the forward and backward paths.

4.2 ULTRA-LOW-PRECISION TRAINING WITH 4-BIT LOGARITHMIC WEIGHTS

Elhoushi et al. (2021) recently demonstrated that 4-bit logarithmic weights could be used in quantized neural networks with minimal accuracy degradation. However, from-scratch training of the same model resulted in a 4.5% accuracy drop in ResNet-18 training (Table 4). Similarly, our experiments show 2.1% lower accuracy when training ResNet-18 using 4-bit logarithmic weights and

Table 4: Comparisons of Training Schemes Using 4-bit Logarithmic Weights for ResNet-18

		Formats (Exponent, Mantissa)			Top-1 Accuracy	
	From Scratch	w	x, dw, dx	Acc.	FP32	Proposed
DeepShift-Q (Elhoushi et al., 2021)	X[1]	(3,0)	-	-	69.8	69.6
	O	(3,0)	-	-	69.8	65.3
FP134 + 4-bit Log W	O	(3,0)	(3,4)	(6,23)	69.8	67.7
FP134 + 4-bit Log W + Hysteresis	O	(3,0)	(3,4)	(6,23)	69.8	69.6

[1] Fine-tuning of pre-trained model

Table 5: 4-bit Logarithmic Weight Training with and without Hysteresis Quantization.

Model (Dataset) [Metric]	FP32	4-bit Log W	4-bit Log W + Hysteresis	Δ
ResNet-18 (ImageNet)	69.8	67.7	69.6	+1.9
MobileNetV2 (ImageNet)	72.2	67.5	69.6	+2.1
2-layer LSTM (PTB) [ppl.]	91.5	93.3	93.2	-0.1[1]
Transformer (IWSLT) [BLEU]	34.8	33.5	33.8	+0.3
MobileNetV2 + SSDLite (VOC) [mAP]	68.3	63.3	66.1	+2.8

[1] Lower is better

FP134 format for other variables. However, using hysteresis quantization greatly improves the training performance, showing only 0.2% accuracy degradation. This is identical to the training performance achieved through fine-tuning a pre-trained model by Elhoushi et al. (2021), confirming that hysteresis quantization effectively solves the issue of sub-optimal solutions in from-scratch training. In addition, Table 5 demonstrates that hysteresis quantization improves the training performance in all target models. Note that we quantized all trainable weights except for the BatchNorm parameters into 4 bits in experiments; the training performance could be further improved by using higher precision for error-sensitive parts such as the first/last layers and residual connections.

We also tested hysteresis quantization on the FP134 training experiments from Section 4.1, but there was no difference in performance across the five models. We suspect that weight fluctuation is less severe in FP134 as it provides a much higher precision than 4-bit logarithmic representation.

5 Conclusion

In low-precision training, the dynamic range of a tensor is data-dependant, and hence an optimal data format depends on various factors such as model, dataset, and quantization scheme. We showed that the training performance of a specific data format could be predicted by observing the errors introduced in the weight gradients. Based on this observation, we could determine an optimal 8-bit format for low-precision training very efficiently without running numerous training runs. The proposed FP134 format achieved a similar or better accuracy compared to prior works, while allowing for efficient hardware implementation through quantizing BatchNorm inputs and using a unified data format in both forward and backward paths. In addition, it was demonstrated that frequent fluctuations in weight parameters hinder reliable training with deeply quantized weights. We proposed the hysteresis quantization scheme, which improves training performance by suppressing undesired weight fluctuations and stabilizing the training process. In ultra-low-precision training with 4-bit logarithmic weights, hysteresis quantization significantly improves training performance by mitigating sub-optimal solutions, closely matching the performance obtained through fine-tuning a pre-trained model.

REFERENCES

Ankur Agrawal, Sae Kyu Lee, Joel Silberman, Matthew Ziegler, Mingu Kang, Swagath Venkataramani, Nianzheng Cao, Bruce Fleischer, Michael Guillorn, Matthew Cohen, et al. A 7nm 4-core ai chip with 25.6 tflops hybrid fp8 training, 102.4 tops int4 inference and workload-aware throttling. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pp. 144–146. IEEE, 2021.

Léopold Cambier, Anahita Bhiwandiwalla, Ting Gong, Mehran Nekuii, Oguz H Elibol, and Hanlin Tang. Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. *arXiv preprint arXiv:2001.05674*, 2020.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, volume 57, 2014.

Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In *MLSys*, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li. Deepshift: Towards multiplication-less neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2359–2368, 2021.

Sean Fox, Seyedramin Rasoulinezhad, Julian Faraone, Philip Leong, et al. A block minifloat representation for training deep neural networks. In *International Conference on Learning Representations*, 2020.

John L Gustafson and Isaac T Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, 2017.

Dong Han, Shengyuan Zhou, Tian Zhi, Yibo Wang, and Shaoli Liu. Float-fix: An efficient and hardware-friendly data type for deep neural network. *International Journal of Parallel Programming*, 47(3):345–359, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

IEEE. Ieee standard for floating-point arithmetic, ieee std 754-2019 (revision of ieee 754-2008). Institute of Electrical and Electronics Engineers New York, 2019.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Urs Köster, Tristan J Webb, Xin Wang, Marcel Nassar, Arjun K Bansal, William H Constable, Oğuz H Elibol, Scott Gray, Stewart Hall, Luke Hornof, et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. *arXiv preprint arXiv:1711.02213*, 2017.

Edward H Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and S Simon Wong. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5900–5904. IEEE, 2017.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *arXiv preprint arXiv:2003.11535*, 2020.

Jeongwoo Park, Sunwoo Lee, and Dongsuk Jeon. A neural network training processor with 8-bit shared exponent bias floating point and multiple-way fused multiply-add trees. *IEEE Journal of Solid-State Circuits*, 2021.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Xiaodong Cui, Wei Zhang, Kailash Gopalakrishnan, et al. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. 2019.

Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7686–7695, 2018.

Yu Emma Wang, Gu-Yeon Wei, and David Brooks. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

Guandao Yang, Tianyi Zhang, Polina Kirichenko, Junwen Bai, Andrew Gordon Wilson, and Chris De Sa. Swalp: Stochastic weight averaging in low precision training. In *International Conference on Machine Learning*, pp. 7015–7024. PMLR, 2019a.

Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xiansheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019b.

A APPENDIX

A.1 VARIOUS FORMATS ANALYZED IN SECTION 2

In this paper, we made three assumptions on the quantization formats that were analyzed. Firstly, 1-bit is allocated as a sign bit, so only symmetric formats are allowed, and secondly, the number of valid bits with a large absolute numerical value must be greater than or equal to the number of valid bits with a small absolute numerical value. Lastly, the base does not exceed 2.

Considering the above assumptions, we provide a systematical approach for generating different quantization methods that were used for analysis in Section 2, in order to create quantization methods that have trade-offs in terms of dynamic range and the number of valid bits. In these quantization methods, we divide the data interval every 2^n (where n is an integer) and define the format by allocating valid bits to each interval. Since the entire interval can be shifted by the scaling factor, the format can be stated by arranging valid bits from the interval with the largest value to the interval with the smallest value without specifying n. The number of intervals to which valid bits are allocated becomes the dynamic range.

When α-bit is given as a format to express data, there are a total of 2^α distinct numerical values that can be expressed in this format. Here, 1-bit is a sign bit, so the positive numerical value that can be expressed is $2^{\alpha-1}$. Now, consider this value as the budget we can use. If the number of valid bits for each interval is selected as m_1, m_2, .., m_β, the following expression must be satisfied.

$$2^{\alpha-1} = \sum 2^{m-1} \tag{7}$$

Since the leading digit is fixed to 1, the number of distinct numerical values required when the valid bit is m is 2^{m-1}. Also, due to the previous constraint, m in the small interval must be less than or equal to m in the large interval. The list of valid bits determined for each interval defines a format, and we denote the valid bits in the order from the number of valid bits in the interval with the largest value to the interval with the smallest value. For example, if 8 bits are given, the number of valid bits can be selected as [7,6,5,4,3,2,1] (which translates to a fixed-point quantization without any exponents) or [6,6,6,5,4,3,2,1]. When various 8-bit data representation formats are expressed as a list of the number of valid bits, it is as shown in Fig. 7.

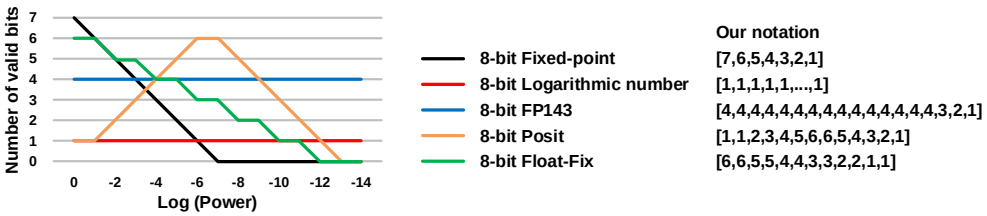


Figure 7: Number of valid bits in various 8-bit formats.

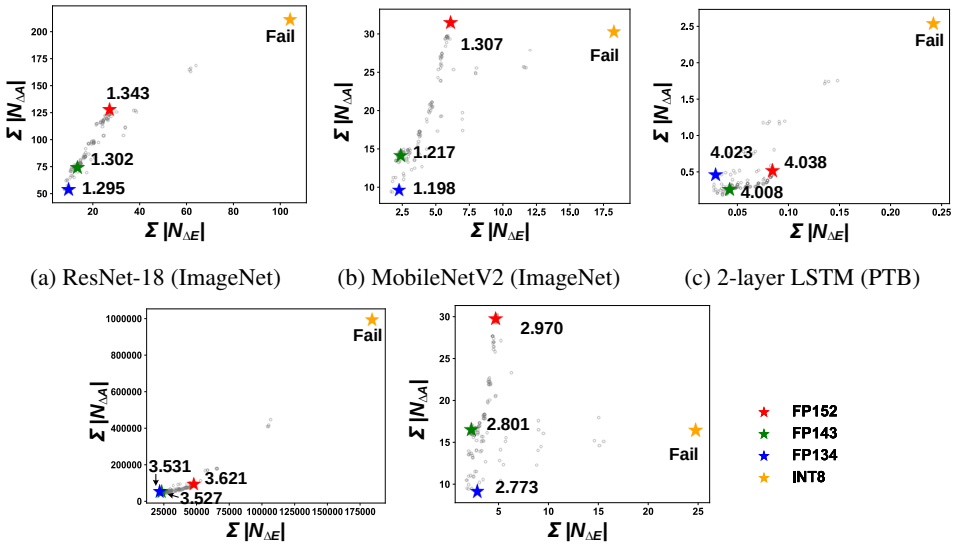
When selecting 8-bit formats, we chose the formats so that the intervals with the number of valid bits that are less than 3 do not appear for more than two digits to reduce the search space, as they have an unnecessarily large dynamic range. Thus, formats such as [7,6,5,4,2,2,2,1] were excluded from the search space. Considering all of the generation rules, we selected 166 distinct 8-bit formats with different dynamic range and valid bits from [7,6,5,4,3,2,1] to [3,3,3,...,3,2,1]. After the number of valid bits for an 8-bit format is selected, 1 or 2 is subtracted from each value to create a corresponding 7-bit and 6-bit formats. For example, in the case of [6,5,5,5,5,4,4,4,3,2,1] 8-bit format, the 7-bit corresponding format is [5,4,4,4,4,3,3,3,2,1] and 6-bit corresponding format is [4,3,3,3,3,2,2,2,1]. From the generated 166 8-bit formats, 7-bit and 6-bit formats were also generated using this rule.

A.2 SOFTWARE IMPLEMENTATION DETAILS

To support quantized training for various formats, custom C++ and CUDA codes to emulate quantized data were written. Our custom C++ and CUDA extension code could perform quantization-related functions through utilizing the Python APIs in PyTorch for extendable research while maintaining high performance. We emulate the quantized value using custom code in the part that needs

quantization throughout the network, and PyTorch built-in functions are used for computation kernels such as convolution and matrix multiplication. We created a package named lptorch, short for low precision PyTorch, and the code can be found in the supplementary material.

A.3 Angle vs. Absolute Value to Predict Performance



(a) ResNet-18 (ImageNet) (b) MobileNetV2 (ImageNet) (c) 2-layer LSTM (PTB)

(d) Transformer (IWLST) (e) MobileNetV2 + SSDLite (VOC)

Figure 8: Scatter plots showing $|N_{\Delta E}|$ and $|N_{\Delta A}|$ of 166 8-bit quantization formats. The number next to the star represents the loss obtained through actual training.

In addition to the misalignment angles of WG ($\angle(WG, WG + N_{\Delta A})$ and $\angle(WG, WG + N_{\Delta E})$), as defined in Section 2.3, we used the absolute values of noise ($|N_{\Delta A}|$ and $|N_{\Delta E}|$) in order to predict the final trained performance, and the results are shown in Fig. 8. In Fig. 8(d), although FP134 shows smaller noise in terms of absolute value, the actual training loss is smaller for FP143. Similarly, in Fig. 8(b) and (e), although INT8 failed and FP152 succeeded in training, the absolute value of noise did not indicate a clear superior of the two formats. Based on these observations, we conclude that the misalignment angles are more suitable for predicting training performance compared against using the absolute value of noise.

A.4 Experimental Details

A.4.1 ResNet-18 (ImageNet)

We conducted ImageNet experiments using SGD with a momentum of 0.9 for 90 epochs with a batch size of 256 images and an initial learning rate of 0.1 which is decayed by a factor of 10 at the 30th and 60th epochs. We used the ResNet-18 architecture from the official PyTorch implementation[1]. Fig. 9 shows Top-1 training & validation accuracy graphs. Observation of the training graph indicates that all of the results are close to the baseline within 0.2% with the exception of FP130 without hysteresis quantization.

A.4.2 MobileNetV2 (ImageNet)

We conducted ImageNet experiments using SGD with a momentum of 0.9 for 270 epochs with a batch size of 256 images and cosine annealing with an initial learning rate of 0.05. We used

[1]https://github.com/pytorch/examples/tree/master/imagenet

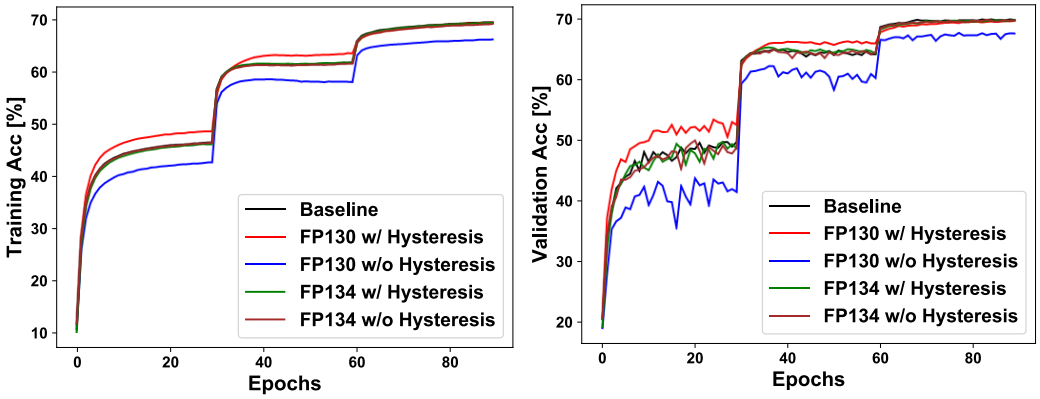


Figure 9: Top-1 Accuracy on ImageNet using a ResNet-18 model.

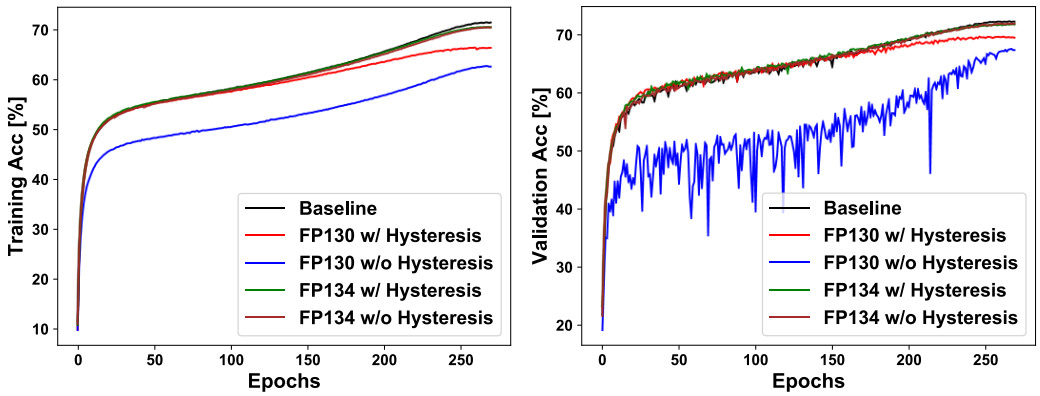


Figure 10: Top-1 Accuracy on ImageNet using a MobileNet-V2 model.

the MobileNetV2 architecture from the official PyTorch implementation[2]. Fig. 10 shows Top-1 training & validation accuracy graphs. Observation of the training graph indicates that FP130 without hysteresis leads to very unstable fluctuations throughout the training. On the other hand, in FP130 with hysteresis, training is less susceptible to fluctuations and follows the baseline (FP32) training closely until the learning rate decreases toward the latter part of learning, where both FP130 with hysteresis and FP134 show some degradation from the baseline. This is seen as a limitation due to the low precision of each format.

A.4.3 2-LAYER LSTM (PTB)

We adopted the 2-layer Long Short Term Memory (LSTM) network from PyTorch Examples[3] for language modeling on the Penn Treebank dataset (Marcus et al., 1993). We ran experiments in batches of 20 sentences with an initial learning rate of 20 which is decayed by a factor of 4 at epoch 11, 16, 26, 31 and 37. The embedding and hidden dimensions are 650 and the sequence length is 35. Fig. 11 shows training & validation perplexity.

A.4.4 TRANSFORMER MODEL (IWLST)

We adopted the Transformer Base model from the FairSeq[4] repository on the IWSLT'14 German to English translation task. We used Adam optimizer and default training parameters found in the

[2]https://github.com/pytorch/examples/tree/master/imagenet
[3]https://github.com/pytorch/examples/tree/master/word_language_model
[4]https://github.com/pytorch/fairseq

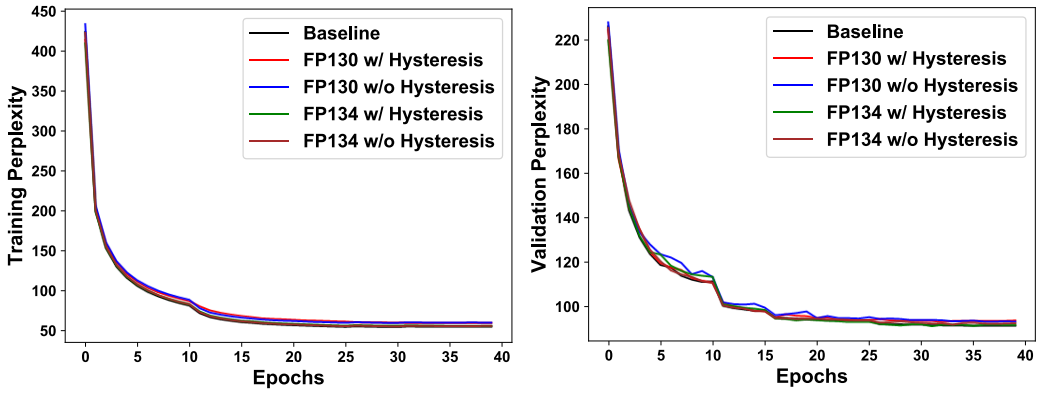


Figure 11: Perplexify on PTB using a 2-layer LSTM model.

repository and trained from scratch for 25 epochs. BLEU scores were calculated using the script from the repository.

A.4.5 MobileNetV2 + SSDLite (VOC)

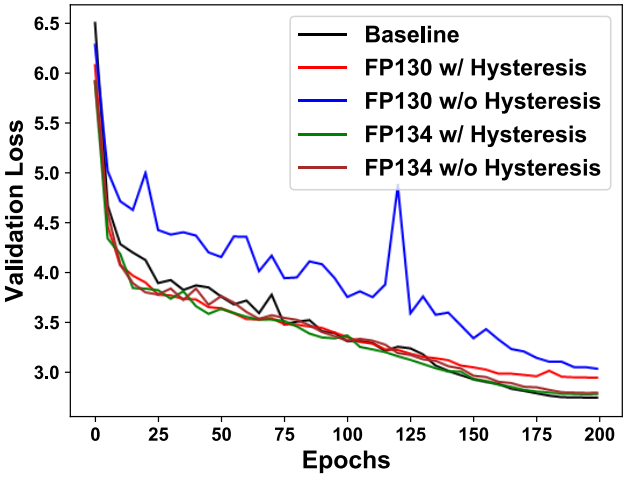


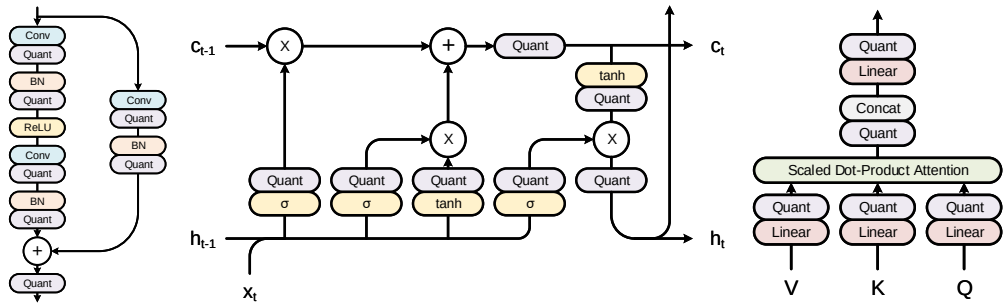
Figure 12: Validation loss on VOC using a MobileNetV2 + SSDLite.

We adopted a PyTorch implementation of SSDLite from the online repository[5]. The base network is MobileNetV2 which was pretrained with each format in Appendix A.4.2. In the case of INT4 without hysteresis, which failed training in MobileNetV2, a model pretrained with INT4 with hysteresis was used. The entire network is trained on VOC2012 and VOC2007 trainval datasets and evaluated on VOC2007 validation dataset. We used SGD with a momentum of 0.9 for 200 epochs in batches of 32 images and cosine annealing with an initial learning rate of 0.01. Fig. 12 shows validation loss at every 5 epochs. Even in this experiment, in the case of FP130 without hysteresis the loss fluctuates significantly, whereas in FP130 with hysteresis learning proceeds much more stably. FP134 showed similar results to the baseline regardless of hysteresis quantization.

A.5 Model Quantization Methods

We quantized GEMM input and batchnorm input in all quantized training experiments. Among the five models used in the experiment, the quantization details for three representative structures are

[5]https://github.com/qfgaohao/pytorch-ssd



(a) Residual Block in ResNet

(b) LSTM

(c) Multi-head Attention

Figure 13: Quantized network structures.

shown in the Fig. 13. In each structure of figure, inputs such as x, c, h, V, K, and Q are also all quantized in 8 bits.