Training Deep Neural Networks in 8-bit Fixed Point with Dynamic Shared Exponent Management

Hisakatsu Yamaguchi
Fujitsu Laboratories LTD.
Kawasaki, Japan
hisakatu@fujitsu.com

Makiko Ito
Fujitsu Laboratories LTD.
Kawasaki, Japan
maki-ito@fujitsu.com

Katsuhiro Yoda
Fujitsu Laboratories LTD.
Kawasaki, Japan
yoda.katsuhiro@fujitsu.com

Atsushi Ike
Fujitsu Laboratories LTD.
Kawasaki, Japan
ike@fujitsu.com

***Abstract*— The increase in complexity and depth of deep neural networks (DNNs) has created a strong need to improve computing performance. Quantization methods for training DNNs can effectively improve computation throughput and energy efficiency of hardware platforms. We have developed an 8-bit quantization training method representing the weight, activation, and gradient tensors in an 8-bit fixed point data format. The shared exponent for each tensor is managed dynamically on the basis of the distribution of the tensor elements calculated in the previous training phase, not in the current training phase, which improves computation throughput. This method provides up to 3.7-times computation throughput compared with FP32 computation without accuracy degradation.**

I. Introduction

Deep neural networks (DNNs) such as AlexNet [1], VGG [2], and ResNet [3] have achieved noteworthy results in image processing and object recognition. In addition, Transformer [4] has also achieved remarkable BLEU (bilingual evaluation understudy) score in machine language translation. In the training of these networks, 32-bit floating point (FP32) or 16-bit floating point (FP16) data representation has been used. However, there is a strong need to reduce the numerical precision of data representation to improve computation throughput and energy efficiency of hardware platforms.

In general, the calculation time of the layer in a DNN includes the time for the calculation itself and the memory access time. The calculation time for matrix multiply (matmul) is mostly dominated by the time of the calculation itself. In contrast, that of vector operation such as AXPY (alpha X plus Y) and batch normalization is mostly dominated by the memory access time. An effective approach to improve computation throughput is therefore to reduce the amount of data for calculation as well as to increase the number of parallel calculations. Quantization can be used to both increase the number of parallel calculations and reduce the amount of data size required for calculations.

We have developed a method for training quantized DNNs that each tensor preserves one shared exponent and elements in an 8-bit fixed point representation. The shared exponent is dynamically calculated on the basis of the distribution of tensor elements calculated in the previous training phase, not in the current training phase, which improves computation throughput. We call this 8-bit fixed point representation with dynamic management of shared exponents (DSE) method "INT8-with-DSE." It provides up to 3.7-times computation throughput compared with FP32 data representation without any degradation in accuracy. In training using INT8-with-DSE, the weight, activation, error and gradient tensors are quantized and represented in an 8-bit fixed point data format. The accuracy achieved is equal to that with FP32 training. In addition, we can apply our method directly to inference devices operating in fixed point.

Two techniques are the keys to achieve this improved performance:

- Dynamic management of shared exponents on the basis of discrete distributions of the tensors calculated in the previous training phase.
- Application of an offset to the shared exponents calculation for deeper networks such as ResNet-50 [3].

In this paper, we describe these techniques and their effect in detail. We also describe the creation of a histogram having base-2 logarithmic bins and showing the discrete distribution of elements constituting each tensor.

II. Related Work

Several works on DNNs training precision scaling have been published over the past few years. A training computation method that uses hybrid 8-bit floating point (FP8) data representation is presented in [5]. It uses a 1-bit sign, 4-bit exponent, and 3-bit mantissa for forward propagation and a 1-bit sign, 5-bit exponent, and 2-bit mantissa for backward propagation. In addition, FP16 representation has also been used to reduce quantization errors when accumulating the results of inner products. A mixed precision technique using both FP8 and FP16 achieved the same accuracy as with FP32 on a wide range of state-of-the art DNN models. The results of the training computation method using floating point, however, indicated that the method suffers from an issue, in that it cannot apply the results obtained through the training directly to inference devices operating in fixed point to realize high energy efficiency. Furthermore, FP8 is not a standard data format and it is required dedicated hardware to support the format.

In contrast, a training computation method using fixed point representation that can be applied to AlexNet and ResNet-18 is presented in [6]. However, the bit width is 16, and there is a strong need to reduce the bit width to improve computation throughput and energy efficiency. In general, the greater the number of layers in a network, the greater its accuracy. The method in [6] and [7] are based on the assumption that the variation in the histogram of elements in each tensor is small between consecutive training phases, so the exponents calculated in the previous training phase can be used in the current training phase. However, the greater the number of layers in a network, the less valid this assumption. This means that the accuracy for ResNet-50 is degraded.

A training computation method that uses block floating point (BFP) representation with an 8-bit mantissa and a 10-bit shared exponent for each tensor and applied it to a ResNet-50 network is presented in [8]. Although the data representation format is the same as in our proposed INT8-

with-DSE method, the method used for calculating the shared exponents differs in terms of improving computation throughput. In both methods, the intermediate data, whose bit width is more than 8, as the results of inner products is calculated, and is converted to 8-bit integer data. In BFP method, the shared exponent to the current training phase is calculated on the basis of the intermediate data, and the conversion using the shared exponent is executed after the calculation. In contrast, in INT8-with-DSE method, the calculation of the shared exponent to the next training phase and the conversion are executed independently, since the shared exponent calculated in the previous training phase is used for the conversion. As a result of the difference in the calculation of the shared exponent, INT8-with-DSE method achieves higher computation throughput than BFP method.

Fig. 1 shows INT8-with-DSE representation and TABLE I. shows a relationship between this work and related work. INT8-with-DSE method is the first 8-bit training method using the shared exponent calculated in the previous training phase, and has potential to achieve the highest computation throughput.

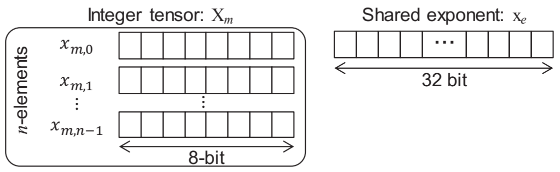


Fig. 1. Diagram of bit representation of INT8-with-DSE.

TABLE I. RELATIONSHIP BETWEEN THIS WORK AND RELATED WORK

		Generation of Shared Exponent	
		Previous Phase	Current Phase
Bit width	8-bit	This Work	[8]
	16-bit	[6], [7]	[9]

III. TRAINING USING INT8-WITH-DSE

A. Data Representation

The INT8-with-DSE method uses a fixed point data format which consists of an integer and an exponent shared among the tensor elements:

$$Quantized(X) = X_m \times 2^{x_e} \tag{1}$$

where X is a tensor whose elements are represented in FP32, and X_m is a tensor whose elements are integers from -128 to $+127$, corresponding to 8-bit integers (INT8). The x_e is the shared exponent [7] of the tensor. The typical histogram of the elements in the weight, activation, and gradient tensors for AlexNet, and the range of values that FP32 and INT8-with-DSE take are shown in Fig. 2. INT8-with-DSE takes a wider range of values due to scaling of the shared exponents dynamically during DNN training and thus can represent all tensors, as shown in the figure. In addition, all tensors obtained through INT8-with-DSE training are represented in fixed point, so inference devices that use fixed point operations are able to handle them directly without conversion from floating to fixed point, as for FP32 and FP16.

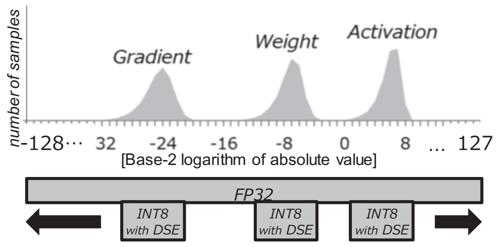


Fig. 2. Histogram of tensor elements for AlexNet, and range of values that can represented in FP32 and INT8-with-DSE.

B. Training Flow and Calculation of Shared Exponents

The INT8-with-DSE training flow is diagrammed in Fig. 3. The training of a DNN comprises a series of training phases, each comprising forward propagation, backward propagation, and weight updating in sequence. The activation tensors for each layer of the network are calculated in the forward propagation step. Similarly, error and gradient tensors for each layer are calculated in the backward propagation step, and weight tensors for each layer are calculated in weight updating step. Shared exponents for each layer are calculated on the basis of a histogram showing the discrete distribution of elements constituting each tensor, as described later, and the calculated shared exponents are used in each layer of the network in the next training phase.

The key feature of the INT8-with-DSE method is the use of the shared exponents calculated in the previous training phase, shown in Fig. 4(a). In INT8-with-DSE method, the results of matmul calculations of 8-bit integer elements in activation tensor and 8-bit integer elements in weight tensor are accumulated as 32-bit intermediate integer data, which is then converted to 8-bit integer data using the shared exponent calculated in the previous training phase. The conversion and the accumulation of the intermediate data as statistics for creating a histogram are executed at the same time.

In contrast, in case of using the shared exponent calculated in the current training phase, as shown in Fig. 4(b), 32-bit intermediate integer data must be temporarily stored in memory to enable it to be converted to an 8-bit integer data since the conversion can be executed only after the shared exponent is calculated. To calculate the shared exponent, the statistics of all intermediate integer data is required. In other words, the conversion, the accumulation of the intermediate data as statistics, and the calculation of the shared exponent must be executed in series.

In addition, the amount of 32-bit intermediate integer data can exceed 100 million, depending on the DNN model. As a result, computation performance is greatly degraded due to limited memory bandwidth if the exponent is calculated in the current training phase. The INT8-with-DSE method thus further improves computation throughput compared with a method using the exponent calculated in the current training phase.

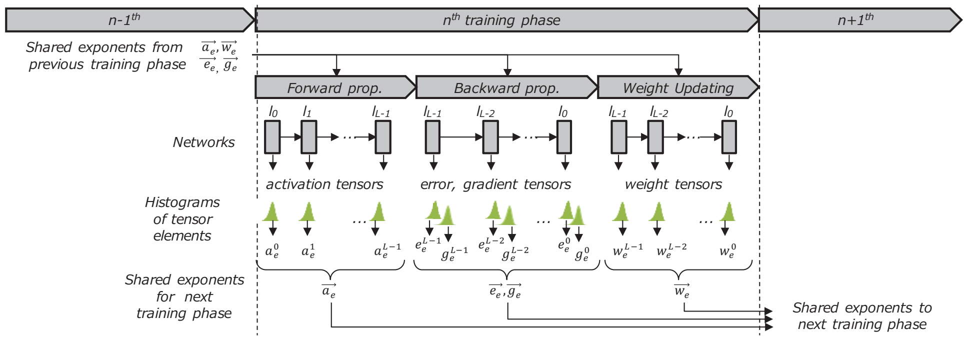
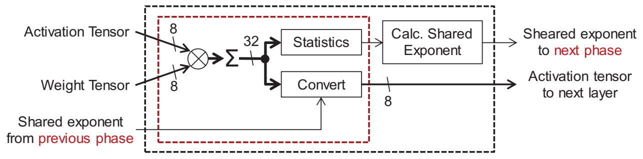
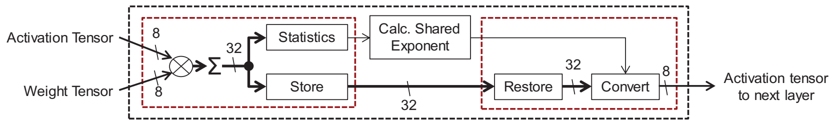

n-1th
nth training phase
n+1th
Shared exponents from previous training phase
Forward prop.
Backward prop.
Weight Updating
Networks
activation tensors
error, gradient tensors
weight tensors
Histograms of tensor elements
Shared exponents for next training phase
Shared exponents to next training phase

Fig. 3. Training flow of INT8-with-DSE.


Activation Tensor
Weight Tensor
Shared exponent from previous phase
Statistics
Convert
Calc. Shared Exponent
Sheared exponent to next phase
Activation tensor to next layer

(a) In case of using shared exponent in previous training phase. (corresponding to INT8-with-DSE)



(b) In case of using shared exponent in current training phase.

Fig. 4. Diagram of matmul operation with shared exponent calculation in previous and current training phase.

C. Matrix and Vector Operation

1) Matrix Multiplication

Fig. 5 shows the calculation flow in matmul operation, the calculation time of which accounts for a large proportion of the total training time. Variables with subscript "e" represent the shared exponent of each tensor. The results of the matmul of the 8-bit integer elements in the activation tensor and the 8-bit elements in the weight tensor are extended to 32-bit fixed point intermediate data. Such matmul operations are generally supported by state-of-the-art processors and are used in inference processing using INT8. Here, the time of matmul operations on INT8 is a quarter of that of FP32.

The shared exponent of the 32-bit fixed point intermediate data t_e^L is the sum of the shared exponent of the activation tensor a_e^L and that of the weight tensor w_e^L. The result of the matmul t_m^L is converted into 8-bit integer A_m^{L+1} on the basis of their shared exponents t_e^L and a_e^{L+1} through Q_{int} as shown in Fig. 5. The conversion formula is given by:

$$Q_{int}(x_m, x_e, y_e) = sat8\big(round(x_m \times 2^{x_e - y_e})\big) \quad (2)$$
$$sat8(x) = max(min(x, 127), -128) \quad (3)$$

where x_m represents input data to Q_{int}, and x_e and y_e represent the input shared exponent and output shared exponent, respectively. When converting the intermediate data t_m^L to 8-bit integer A_m^{L+1}, x_e and y_e correspond to t_e^L and a_e^{L+1}, respectively.

The *round* function in (2) corresponds to stochastic rounding [10]. The sat8 function in (3) truncates the input value to 127, which is the maximum value expressed by an 8-bit integer if it is more than 127. The function also truncates it to -128, which is the minimum value expressed by an 8-bit integer if it is less than -128.

2) Vector Operations

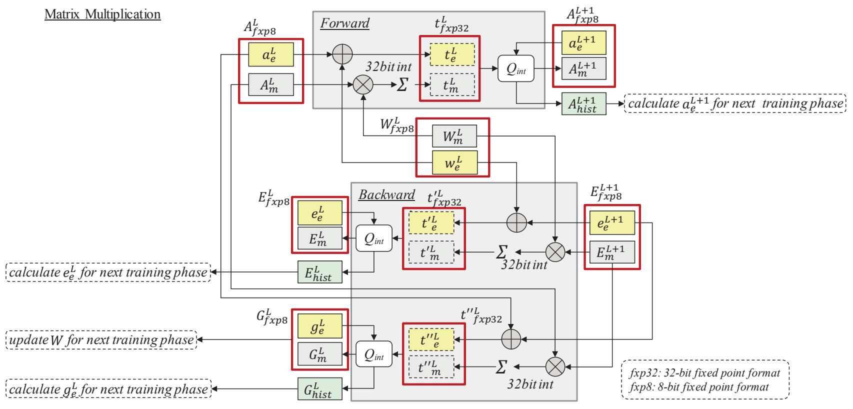
Fig. 6 shows the calculation flow in a set of vector operations such as one or a set of AXPY operations and batch normalization. In this flow, the shared exponents calculated in the previous training phase are used.

Although the vector operation is executed in 32-bit floating point so as not to degrade accuracy, the bit width of the input and output tensor is 8, not 32, since the computation throughput of vector operation is limited by memory bandwidth. The conversion formula for the bit width at the input and output nodes is:

$$to\_float(x_m, x_e) = x_m \times 2^{x_e} \quad (4)$$
$$to\_fixed\ (x, x_e) = round(x \times 2^{-x_e}) \quad (5)$$

3) Weight Updating

Accuracy degradation due to quantization can be effectively suppressed by updating the weights in full FP32 precision [11], and this approach is used in INT8-with-DSE training, as shown in Fig. 7. The updating weights are calculated using weights and gradients converted into FP32, and then, the updating weights are converted to 8-bit integer. During this conversion, the gradient is reflected in the updating weights by applying stochastic rounding.



Matrix Multiplication
Forward
Backward
32bit int
calculate a_e^{L+1} for next training phase
calculate e_e^L for next training phase
update W for next training phase
calculate g_e^L for next training phase
fxp32: 32-bit fixed point format
fxp8: 8-bit fixed point format

Fig. 5. Data flow diagram of matmul operation during forward and backward paths.

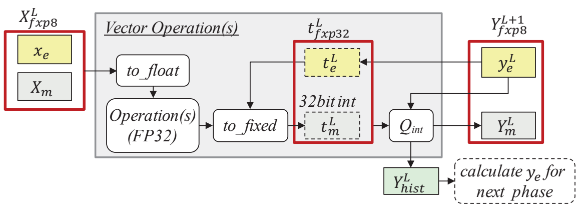


Fig. 6. Data flow diagram of vector operation.

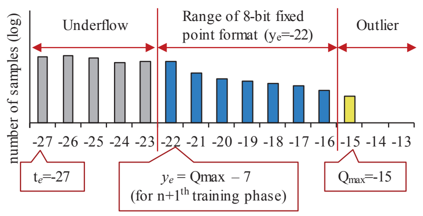


Fig. 8. Histogram Y_{hist} of the n^{th} training phase.

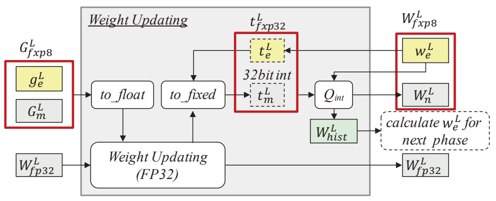


Fig. 7. Data flow diagram for weight updating.

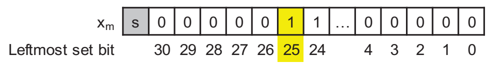


Fig. 9. Diagram of leftmost set bit of integer value.

D. Histogram Creation and Shared Exponent Calculation

1) Histogram Creation

A histogram describing the discrete distribution of tensor elements is created from the 32-bit intermediate results t_m^L in Fig. 5 to 7, as shown in Fig. 8. To create a histogram which has base-2 logarithmic bins, $\lfloor log_2(t) \rfloor$ is calculated using

$$\lfloor log_2(t) \rfloor = \lfloor log_2(t_m \times 2^{t_e}) \rfloor \quad (6)$$
$$= \lfloor log_2(t_m) \rfloor + t_e \quad (7)$$

where t denotes an intermediate result, consisting of an integer part t_m and a shared exponent t_e. From (7), $\lfloor log_2(t) \rfloor$ can be calculated by adding $\lfloor log_2(t_m) \rfloor$ and t_e. For integer data t_m, calculating $\lfloor log_2(t_m) \rfloor$ is equivalent to getting the position of the leftmost set bit from the hardware point of view, as shown in Fig. 9.

The position can be calculated by several instructions, such as "count leading sign" instruction. The increase in computation time due to creating a histogram can be ignored, since computation time of matmul operation is dominant. For vector operation, the increase in computation time due to creating a histogram can also be ignored, since the calculation time of the layer is dominated by the memory access time.

2) Shared Exponent Calculation

An example histogram is shown in Fig. 8. Shared exponent y_e is calculated by, first, determining Q_{max} so that any outliers, which is shown in yellow in Fig. 8, are removed. Here, we define r_{max} as the outlier rate, which corresponds to the overflow rate in [7]. Q_{max} represents the maximum bit obtained by removing the outlier corresponding to r_{max} from the maximum index of the histogram, as shown in Fig. 8. Then, y_e is calculated by subtracting bit-width excluding sign bit, 7 for INT8-with-DSE, from Q_{max}.

On the assumption that the distributions of tensor elements between consecutive training phases are similar, shared exponent y_e for the $n+1^{th}$ training phase is determined from the histogram of the n^{th} training phase. However, in deeper networks, the histogram of each tensor shows a larger variation across the training phase, so the shared exponent in the current training phase cannot be predicted accurately from the previous training phase, as shown in Fig. 10. From the distribution for the n^{th} training phase shown in Fig. 8, shared exponent y_e is set to -22. In the $n+1^{th}$ training phase, the values of tensor Y increase and the distribution moves to right. As a result, the ratio of overflow data has exceeded the r_{max}. Values are unexpectedly overflow, as shown by the red bar in Fig. 10, and the precision of the data is degraded with shared exponent $y_e = -22$. To prevent such overflow, we add an "offset" to the shared exponent. Then, y_e is thus calculated from Q_{max} and "offset":

$$y_e = Q_{max} - 7 + offset \quad (8)$$

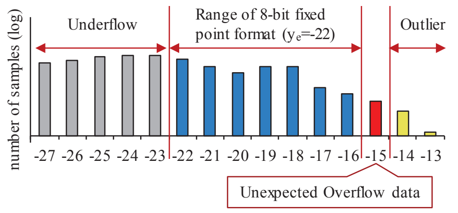


Fig. 10. Histogram Y_{hist} of the $n + 1^{th}$ training phase. From distribution variation, shared exponent y_e is not predicted accurately.

IV. Experiment and Result

A. *Image classification*

To evaluate the accuracy of image classification DNNs quantized with INT8-with-DSE, we trained AlexNet [1], VGG19 [2], and ResNet-50 [3] with INT8-with-DSE using ImageNet. For the acceleration of training, we inserted batch normalization layer [12] to VGG19, and we call this VGG19 model "VGG19BN". We also compared the accuracy of the quantized DNNs against those of the DNNs trained with FP32. The hyperparameters used in this experiment are shown in TABLE II. The learning rate was reduced from the initial learning rate in one step for AlexNet or in two steps for the VGG19BN and ResNet-50 networks. Momentum Stochastic Gradient Descent (mSGD) [13] was also applied for improving accuracy.

TABLE II. Hyperparameter settings in experiment

Parameter	AlexNet	VGG19BN	ResNet-50
Mini-batch size	256	32	256
Dropout rate	0.5	0.5	-
Optimizer	mSGD	mSGD	mSGD
Momentum	0.9	0.9	0.9
Weight decay*	0.0005	0.0005	0.0001
Regularization type	L2	L2	L2
r_{max} for DSE	0.01%	0.01%	0.002%

*Default values are based on the vanilla networks.

1) *Date Format For Layers*

We applied mixed precision with INT8-with-DSE and FP32. INT8-with-DSE was applied to all layers excluding the last fully connected layer and softmax layer for all three networks. For ResNet-50, in addition that, we excluded to apply INT8-with-DSE to batch normalization, ReLU and max pooling layers in the first conv1 block.

2) *Offset for Shared Exponent*

The variation in the histogram of each tensor in each training phase for AlexNet and VGG19BN networks is relatively small since the number of layers is small. As a result, the shared exponents calculated in the previous training phase can be directly used for the calculation in the current training phase. For deeper networks, however, such as ResNet-50, the variation in the histogram of each tensor is larger across the training phase, so the shared exponents in the current training phase cannot be accurately predicted from the ones calculated in the previous training phase.

Fig. 11 shows the transition in the difference between the shared exponent calculated on the basis of the histogram of current training phase and previous training phase for error tensors in ResNet-50. The x-axis of the heatmap shows the training epochs and the y-axis denotes the difference. The cell of the heatmap indicates the cumulative value of each difference among 5 epochs.

Fig. 11 shows that the lower the learning rate, the smaller the difference. Therefore, we adjusted the offset in accordance with the learning rate. Controlling the offset values of the shared exponents resulted in accuracy for ResNet-50 equivalent to that for FP32 calculations. Furthermore, the offset was adjusted to decrease as the learning rate decreased.

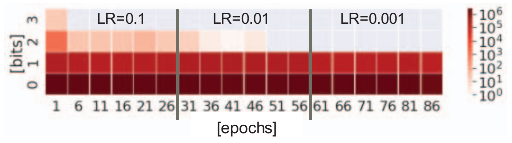


Fig. 11. Transition of shared exponent difference for backward propagation path of ResNet-50.

3) *Throughput*

We estimated training time on each DNN. In each layer of DNNs, the computation and memory access to obtain the data required for the computation are executed in parallel. We estimated both of computation time and memory access time for each layer, and chose the longer time from the two. The computation time was estimated based on the total operation count and the peak computation performance as shown in TABLE III. The memory access time was also estimated based on the total input and output tensor size and the memory throughput as shown in TABLE III. The memory throughput of INT8-with-DSE is four times greater than that of FP32. The performances in TABLE III. correspond to those of readily available GPU [14].

The estimated training time on ResNet-50 is shown in Fig. 12. The estimated training time for INT8-with-DSE includes 10 additional operation time per output tensor element as an overhead of creating a histogram. Our estimation is that the computation throughput is improved up to 3.7-times over that of FP32 by using INT8-with-DSE. We estimated computation throughput in the same manner in regards with AlexNet and VGG19BN, the computation throughput is improved up to 3.2 and 3.4-times over that of FP32 by using INT8-with-DSE, respectively.

TABLE III. Comparison performance and memory bandwidth for training time estimation

	Computation(operation/s)	Memory(element/s)
FP32	15T	225G
INT8	60T	900G

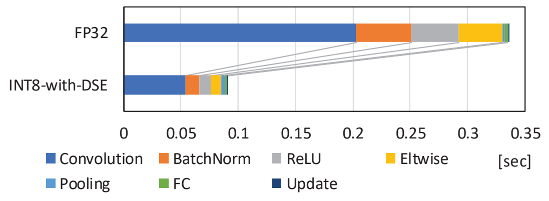


Fig. 12. Graph of estimated training time per training phase of ResNet-50.

4) *Results of Image Classification*

The experimental results are shown in Fig. 13 and TABLE IV. The accuracy of the DNNs quantized with INT8-with-DSE was equal to that with FP32.

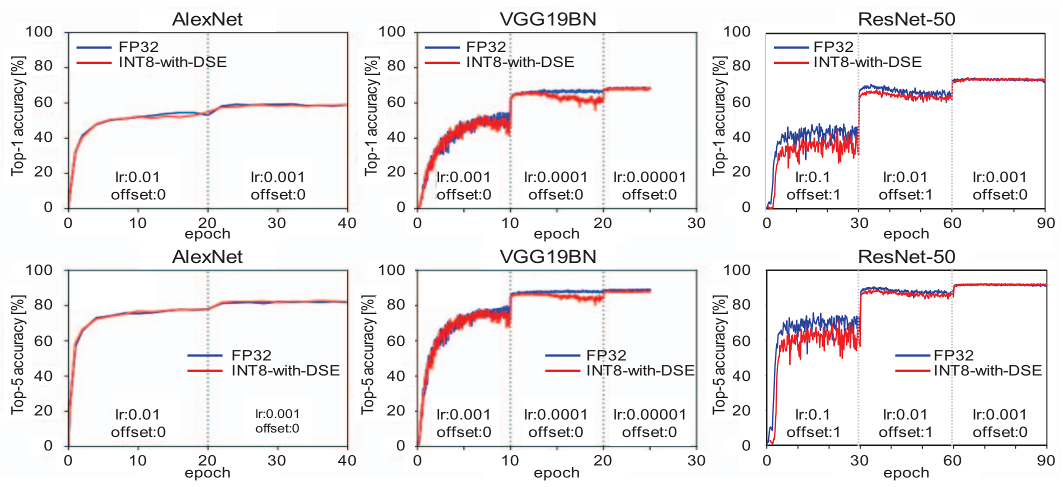


Fig. 13. Top-1 and top-5 accuracies of evaluated networks and each "offset" setting. "lr" denotes the learning rate.

TABLE IV. TOP-5 ACCURACY AND PERFORMANCE IMPROVEMENT RATIO FOR INT8-WITH-DSE COMPARED WITH FP32

	AlexNet	VGG19BN	ResNet-50
FP32	79.44%	89.22%	92.28%
INT8-with-DSE	79.28%	88.59%	92.22%
Throughput	3.2x	3.4x	3.7x

B. *Natural language processing*

We also trained a 6-layer Transformer-base translation network on WMT17 English to German dataset. The matmul calculations, except for the two linear transformations in the fully connected feed-forward networks in the encoder and the linear transformation in the decoder output, were executed using INT8-with-DSE (r_{max} = 0% and offset=0). The distribution of the activation tensor elements of the linear transformations to which INT8-with-DSE could not be applied were not in 8-bit range, so the calculation of these linear transformations was executed in INT16-with-DSE. As shown in TABLE V. , the BLEU score quantized using INT8/16-with-DSE was equal to that with FP32. The estimated computation throughput is improved up to 2.1-times over that of FP32 using INT8/16-with-DSE.

TABLE V. BLEU SCORE AND THROUGHPUT FOR TRANSFORMER

	Score	Throughput
FP32	23.95	1.0
INT8/INT16-with-DSE	23.97	2.1

V. Conclusion

In our proposed 8-bit training method for quantized DNNs, the distributions of the tensor elements calculated in the previous training phase are used in the current training phase. The weight, activation, and gradient tensors are represented in an 8-bit fixed point data format, and the shared exponent are dynamically controlled. Satisfactory accuracy for deeper networks such as ResNet-50 was obtained by adjusting the offset added to the shared exponents in accordance with the variation in the statistics. The proposed quantization method can provide up to 3.7-times computation throughput compared with FP32, without degrading the accuracy of the DNN.

References

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] Simonyan, K. and Zisserman, A., " Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I., "Attention Is All You Need," *Advances in neural information processing systems,* pp. 5998-6008, 2017.

[5] Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V. V., Cui, X., Zhang, W., and Gopalakrishnan, K., "Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks," *Advances in Neural Information Processing Systems 32,* pp. 4901–4910. Curran Associates, Inc., 2019.

[6] Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A. K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L., et al., "Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks," *Advances in Neural Information Processing Systems,* pp. 1742–1752, 2017.

[7] Courbariaux, M., Bengio, Y., and David, J.-P., "Training Deep Neural Networks with Low Precision Multiplications," *arXiv preprint arXiv:1412.7024,* 2014.

[8] Drumond, M., Tao, L., Jaggi, M., and Falsafi, B., "Training DNNs with Hybrid Block Floating Point," *Advances in Neural Information Processing Systems,* pp. 451–461, 2018.

[9] Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., Heinecke, A., Dubey, P., Corbal, J., Shustrov, N., Dubtsov, R., Fomenko, E., and Pirogov, V., "Mixed Precision Training of Convolutional Neural Networks using Integer Operations," *arXiv preprint arXiv:1802.00930,* 2018.

[10] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P., "Deep Learning with Limited Numerical Precision," *International Conference on Machine Learning,* pp. 1737– 1746, 2015.

[11] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaev, O., Venkatesh, G., et al., "Mixed Precision Training," *arXiv preprint arXiv:1710.03740,* 2017.

[12] Ioffe, S. and Szegedy, C., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv preprint arXiv:1502.03167,* 2015.

[13] Qian, N., "On the momentum term in gradient descent learning algorithms," *Neural networks,* 12(1):145–151, 1999.

[14] https://www.nvidia.com/en-us/data-center/v100/