修士論文

Optimization of Numerical Expression in CNN
(CNN における数値表現の最適化)

平成 31 年 1 月 30 日提出

指導教員
工藤　知宏　　教授

電気系工学専攻　融合情報コース
37-176471　野上　和加奈

Abstract

The size and complexity of CNN models are increasing and as a result they are requiring more computational and memory resources to be used effectively. Use of a lower bit width numerical representation such as binary, ternary or several bit width has been studied extensively so as to reduce the required resources. However, the representation capability of such extremely low bit width is not always sufficient and the accuracy obtained for some CNN models and data is low. There are some prior studies that use moderate lower bit width with well-known numerical representations such as fixed point or logarithmic representation. It is not apparent, however, whether those representations are optimal for maintaining high accuracy. In this thesis, I investigated the numerical quantization from the ground up, and introduced a novel "Variable Bin-size Quantization (VBQ)" representation in which quantization bin-sizes are optimized to obtain maximum accuracy for each CNN model. A genetic algorithm was employed to optimize the bin-sizes of VBQ. Additionally, since the appropriate bit width to obtain sufficient accuracy cannot be determined in advance, I used the parameters obtained by a training process using higher precision representation (FP32), and used quantization in inference only. This reduced the required large computational resource cost for training. During the process of tuning VBQ bin-sides using a genetic algorithm, I discovered that the optimal distribution of bins can be approximated by an equation with two parameters. I then used simulated annealing for finding the optimal parameters of the equation for AlexNet and VGG16. As a result, AlexNet and VGG16 with 4-bit quantization achieved top-5 accuracy at 74.8% and 86.3% respectively, which were comparable to 76.3% and 88.1% obtained by FP32. Thus, VBQ combined with the approximate equation and the simulated annealing scheme can achieve similar levels of accuracy with less resources and reduced computational cost compared to other current approaches.

CONTENTS

LIST of FIGURES

LIST of TABLES

Chapter 1

Introduction

1.1 Background

Convolutional Neural Network (CNN) is widely used in various tasks such as image recognition. In order to improve recognition accuracy, it is necessary to use the models which have many layers. However, the increasing size and complexity of CNN models is requiring more computational power and memory. Some of the larger models can only be run on limited high-end servers [4, 5, 6]. For example, ResNet-50 takes 29 hours to train ImageNet using 8 Tesla P100 [7]. In the latest research, Mikami et al. have conducted the same training in 224 seconds, but they used up to 2,176 Tesla V100 [6]. However, not all peple can prepare sufficient environments. Thus, those people often use pri-trained models and run only inference, but it is still too complex to run on edge devices or small servers. There is a demand for executing inference on edge devices or small servers because it is sometimes undesirable to use data centers or supercomputers due to security and response time issues. Thus, it is necessary to compress the complex model to adapt to those environments.

Compression techniques to reduce the trained model size are extensively studied. Reducing bit width in quantizing values is one of the promising approaches for the compression. With the reduced bit width, both computation and memory footprint are expected to be reduced. It is even possible to save energy for inference, by preparing specially tailored hardware. There are two different approaches to bit width reduction: one includes training and the other does not. In the former approach, a quantized model is built from scratch, which is then trained [8, 9, 10, 11, 12]. With this approach, improved accuracy may be achieved compared to the corresponding floating point 32bit (FP32) results [11], and the compression rate is generally high. However, building and tuning the new models require a lot of effort, and are time consuming. The latter approach aims at reducing the bit width of already-trained models without affecting their accuracy [13, 14, 15]. Although this approach is difficult because quantization error can accumulate, existing models with established trained data can be reused without the costly training process.

This work described in this paper focuses on the second approach. Different from prior

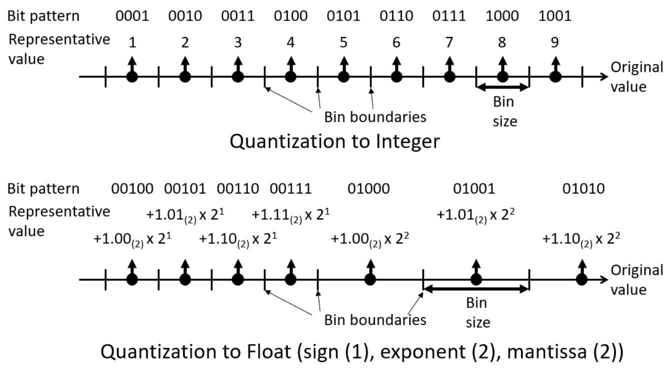


Figure.1.1: Quantization examples

studies, I made no assumption about the quantization format, such as fixed point, floating point, or logarithmic representations. Instead, I find an optimal quantization under a given bit width. The quantization can be defined as mapping a continuous value onto a numerical representation of finite bit width. In more detail, a continuous value range is divided into bins (sections) indexed by the given bit width, and any numbers in each bin are replaced by a representative value assigned to the bin. Quantization schemes are depicted in Figure 1.1 for two widely used representations: the fixed point format and the floating point format. These schemes are designed to have systematically determined bin sizes and representative values, and to be easily processed mathematically. However, they are not necessarily optimal in maximizing the accuracy of CNN under the given bit width. By arranging bin boundaries and representative values more flexibly, it will be possible to improve the accuracy, or to reduce the bit width further with keeping the accuracy.

1.2 Contributions

The contributions of this research are as follows:

- I proposed a scheme called variable bin size quantization (VBQ) and applied it to CNN.
- The bin sizes (or the representative values) are optimized by using the genetic algorithm.
- I derived a formula to describe the distribution of the representative values, which represent continuously both the fixed point-like quantization and the floating point-like quantization.
- I optimized parameters in the formula using simulated annealing and succeeded in reducing the representation of the weight of all layers to 4 bits with only a slight reduction in top-5 accuracy of 1.84% in AlexNet and 1.50% in VGG16 respectively.
- I found that in order to keep the accuracy drop less than 1% compared with FP32, the activation of input layer requires 4 bits, but the activation of other layers requires 1 bit at the minimum and 3 bits at the maximum.

1.3 Structure of Thesis

This thesis is organized as follows: In Chapter 2, I present an extensive literature survey of the related works. In Chapter 3, I develop a CNN framework and investigate weights in CNN. In Chapter 4, I optimize quantization of weights using a genetic algorithm and then optimize parameters for quantization using a simulated annealing in Chapter 5. In Chapter 6, I investigate and experiment activation in CNN. It is followed by conclusions in the last chapter.

Chapter 2

Related Work

2.1 Convolutional Neural Networks

In this thesis, I target on convolutional neural networks(CNN). First, I explain the basic structure and terms of CNN. A convolutional neural network is a neural network having a convolutional layer, and is generally used in image recognition. The architecture example is shown in Figure.2.1. The value propagating through each layer from input to output is defined as Activation; the value multiplying the activation in the convolutional layer or the fully connected layer is called Weight. Receiving an image as input, predicting what the subject is and outputting it as a probability is called inference. The process of comparing inference results with correct answers, propagating the error from output to input, and updating weights is called training. Also, the value propagating from the output toward the input is called Gradient.

MNIST, CIFAR-10, ILSVRC2012 are commonly used data sets for CNN research. MNIST is a data set of 10 class handwritten numbers from 0 to 9 and the color is gray scale and the image size is 28 * 28. It is one of the simplest data sets because it has a small number of classes, only one gradation in color, and small image size. CIFAR-10 is a 10-class data set of vehicles and animals and the color is three channels of RGB and the image size is 32 * 32. It is a data set that is more difficult than MNIST because it has three color channels of RGB. ILSVRC

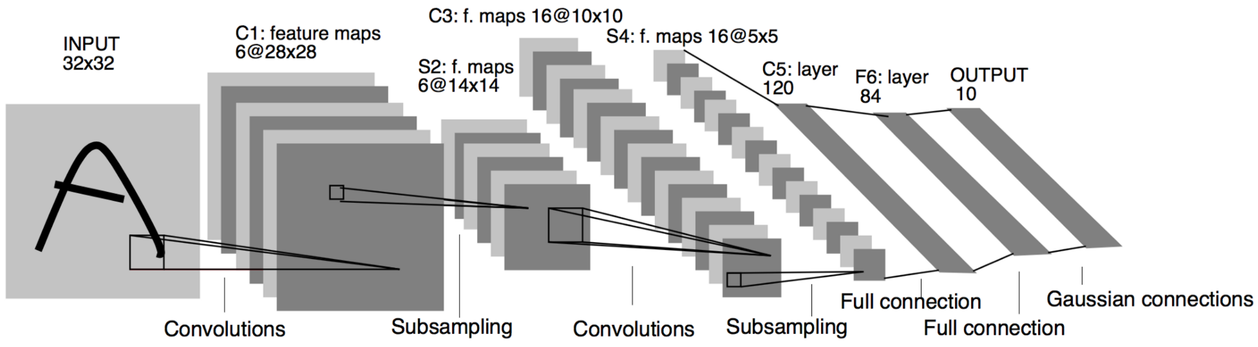
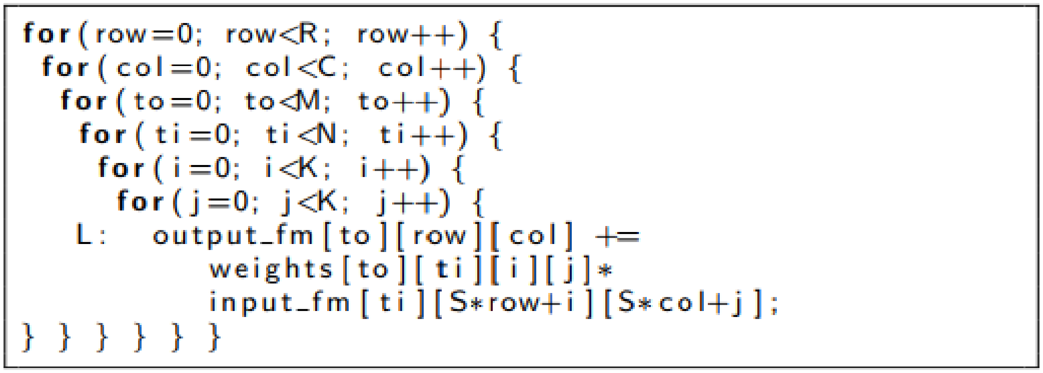


Figure.2.1: Architecture example of CNN [1]



```
for(row=0; row<R; row++) {
 for(col=0; col<C; col++) {
  for(to=0; to<M; to++) {
   for(ti=0; ti<N; ti++) {
    for(i=0; i<K; i++) {
     for(j=0; j<K; j++) {
   L:   output_fm[to][row][col] +=
           weights[to][ti][i][j]*
           input_fm[ti][S*row+i][S*col+j];
} } } } } }
```

Figure.2.2: The calculation of convolutional layer. [2]

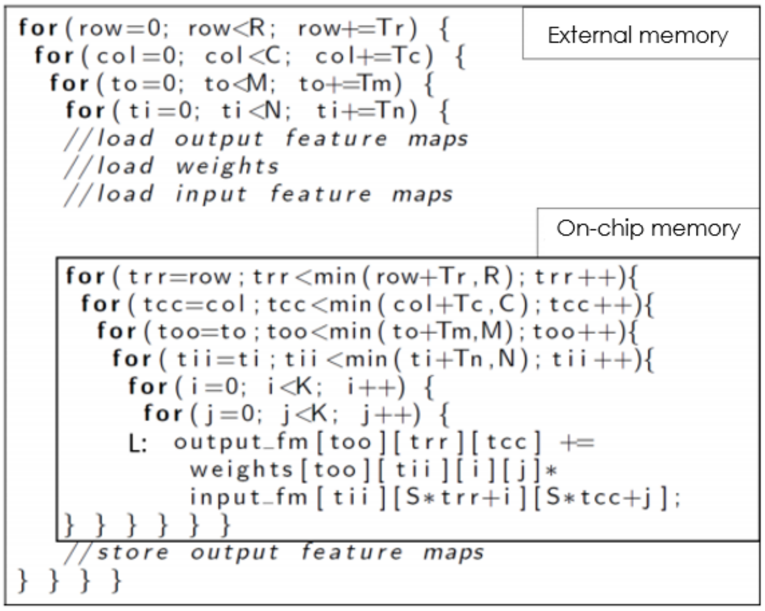
2012, which is sometimes called just ImageNet, is a data set used in the 2012 challenge of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which is an image recognition competition using ImageNet. ILSVRC 2012 is a data set of 1000 class general object. It has 3 color channels of RGB and the image size is 224 * 224 or 227 * 227. It is the most difficult data set because the image size and number of classes are large. Also, commonly used network constructions are named, such as LeNet [1], AlexNet [16], VGG16 [17], GoogLeNet [18] and ResNet [7]. In this thesis, I call the network construction as Model.

2.2 Acceleration on FPGA

The final objective of this research is to reduce the number of bits of numerical expression in order to make calculations based on LUT (look up table) of FPGA, and to accelerate the calculation of deep learning. I investigated the related works about the acceleration method for deep learning using FPGA.

2.2.1 Loop Optimization

Operations in convolution layers occupy more than 90% of the operations in CNN [19]. It is very important to shorten the computation time of the convolution layer in order to improve the efficiency of processing. I showed the calculation of convolutional layer in Figure.2.2 In Figure.2.2, K is a kernel size, N is the input channel size, M is the output channel size, and R and C is the number of row and column of output respectively. The operation when this calculation is performed on the FPGA is as follows.



```
for(row=0; row<R; row+=Tr) {
 for(col=0; col<C; col+=Tc) {
  for(to=0; to<M; to+=Tm) {
   for(ti=0; ti<N; ti+=Tn) {
    //load output feature maps
    //load weights
    //load input feature maps
    for(trr=row;trr<min(row+Tr,R);trr++){
     for(tcc=col;tcc<min(col+Tc,C);tcc++){
      for(too=to;too<min(to+Tm,M);too++){
       for(tii=ti;tii<min(ti+Tn,N);tii++){
        for(i=0; i<K; i++) {
         for(j=0; j<K; j++) {
        L:  output_fm[too][trr][tcc] +=
              weights[too][tii][i][j]*
              input_fm[tii][S*trr+i][S*tcc+j];
    } } } } } }
    //store output feature maps
} } } }
```

Figure.2.3: Loop Tiling [2]

1. Load the data necessary for computation from the external memory into the on-chip memory
2. Do calculation
3. Load again from the external memory if the data necessary for computation is not on the chip

Since loading data from the external memory to the on-chip memory is very time consuming, it is important to reduce the number of data loading for efficiency. Loop tiling is the one of the method for reducing the number of data loading. Loop tiling, it is also called loop blocking, is a method to make memory access more efficient by dividing the loop into small blocks. Tiling the loop of convolution layer represented by Figure.2.2 results in Figure.2.3. Unnecessary memory access such as reloading the same data over and over can be eliminated by dividing the loop into the sizes which all data required for one calculation falls in the on-chip memory. As a result, the number of times of loading from the external memory is reduced, and the calculation can speed up. Since the tile size of loop tiling is related to the number of computing units to be used and the number of times of access to external memory, it is an important parameter for efficiency improvement and needs to be carefully decided.

Zhang et al proposed a method of determining the tile size at loop tiling using a roof line model [2]. The roof line is a model showing the upper limit to the computation performance of the program depending on the data transfer amount (Figure.2.4).

The left part of the graph of Figure.2.4 means when the ratio of operations to data transfer is small, the maximum number of computation per unit time is limited by the bandwidth of data transfer. The flat part of the graph means when the ratio of the data transfer becomes larger

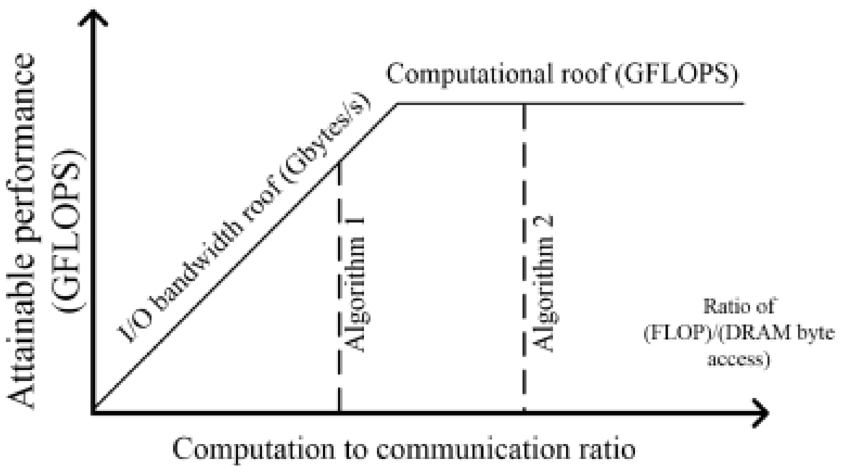


Figure.2.4: Roof Line Model [2]: The y axis represents the ratio of calculation to data transfer and the x axis represents calculation performance.

```
//on-chip data computation
for(i=0; i<K; i++) {
 for(j=0; j<K; j++) {
  for(trr=row; trr<min(row+Tr,R); trr++){
   for(tcc=col; tcc<min(col+Tc,C); tcc++){
#pragma HLS pipeline
    for(too=to; too<min(to+Tm,M); too++){
#pragma HLS UNROLL
     for(tii=ti; tii<min(ti+Tn,N); tii++){
#pragma HLS UNROLL
      L:  output_fm[too][trr][tcc] +=
            weights[too][tii][i][j]*
            input_fm[tii][S*trr+i][S*tcc+j];
} } } } } }
```

Figure.2.5: Convolutional Layer with Tiling, Unrolling and Pipe-lining [2]

than a certain value, the bottleneck of data transfer disappears and the maximum number of the calculation per unit time becomes the same as the theoretical calculation performance.

Figure.2.5 is an computation example when the tiling size of N, M, R and C is T_N, T_M, T_R and T_C respectively. It is also applied loop unrolling and loop pipe-lining. I can compute in parallel by expanding the loop since there are multiple multipliers/adders in FPGA. By computing in parallel, it is possible to maximize the use of the resources of FPGA and to improve the efficiency of processing. Also, the number of operations that can be executed at a time is limited by the number of resources, but it can be executed at the same time as long as it uses different resources. It is possible to execute different operations simultaneously by pipe-lining and maximize the use of all resources. In Figure.2.5, the computational roof, which represents the theoretical computational upper limit, can be calculated as follows.

$$
\begin{aligned}
& Computation roof \\
& = \frac{total number of operations}{number of execution cycles} \\
& = \frac{2 \times R \times C \times M \times N \times K}{\lceil \frac{M}{T_M} \rceil \times \lceil \frac{N}{T_N} \rceil \times \frac{R}{T_R} \times \frac{C}{T_C} \times (T_R \times T_C \times K \times K + P)} \\
& \approx \frac{2 \times R \times C \times M \times N \times K}{\lceil \frac{M}{T_M} \rceil \times \lceil \frac{N}{T_N} \rceil \times R \times C \times K \times K} \\
& where \\
& \quad P = pipelinedepth - 1
\end{aligned}
\tag{2.1}
$$

Then, Computation to Communication Ratio, which represents the ratio of operations to data transfer, can be calculated as follows.

$$
\begin{aligned}
& Computation to Communication Ratio \\
& = \frac{total number of operations}{total amount of external data access} \\
& = \frac{2 \times R \times C \times M \times N \times K}{\alpha_{in} \times B_{in} + \alpha_{weight} \times B_{weight} + \alpha_{out} \times B_{out}} \\
& where \\
& \quad B_{in} = T_N(ST_R + K - S)(ST_C + K - S) \\
& \quad B_{weight} = T_M T_N K^2 \\
& \quad B_{out} = T_M T_R T_C \\
& \quad 0 < B_{in} + B_{weight} + B_{out} \leq BRAM_{capacity} \\
& \quad \alpha_{in} = \alpha_{weight} = \frac{M}{T_M} \times \frac{N}{T_N} \times \frac{R}{T_R} \times \frac{C}{T_C} \\
& \quad \alpha_{out} = \frac{M}{T_M} \times \frac{R}{T_R} \times \frac{C}{T_C}
\end{aligned}
\tag{2.2}
$$

M, N, R, C, K, S is defined by a model, so the computation roof and the computation to communication ratio are changed depending on T_R, T_C, T_M, T_N. Those parameters are restricted by following equations.

$$
\begin{cases}
0 < T_M \times T_N \leq (\# \quad of \quad PEs) \\
0 < T_M \leq M \\
0 < T_N \leq N \\
0 < T_R \leq R \\
0 < T_C \leq C
\end{cases}
\tag{2.3}
$$

The result of variously changing T_R, T_C, T_M, T_N while satisfying those equations is shown in Figure.2.6. Please note that design A in Figure.2.6 can only demonstrate the performance of A' due to data transfer time constraints. Therefore, design C and D are more effective than

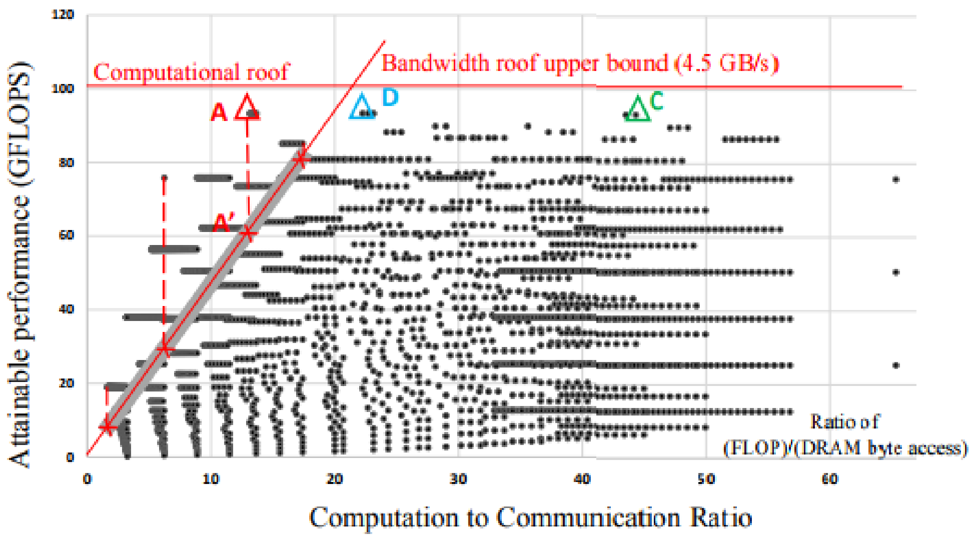


Figure.2.6: Design Space of All Possible Designs [2]

design A. The design which has the larger ratio of operation to transfer is suitable for FPGA since the FPGA generally has a narrow bandwidth. Thus, in this case the design C is most appropriate. As a result of deciding the optimal design with this method, the processing speed was at least 3.62 times faster than the other FPGA implementation. Moreover, compared with the implementation on 16 threads CPU, the speed was 4.8 times and the power consumption was 1/24.6 times.

2.2.2 Fixed-Point Representation

Fixed-point is a represent method with only the mantissa part by fixing the digit to which the decimal point is placed. Compared with floating-point representation, there is an advantage that calculation is simple. The disadvantage is that the quantization error is large and it leads to decrease an accuracy.

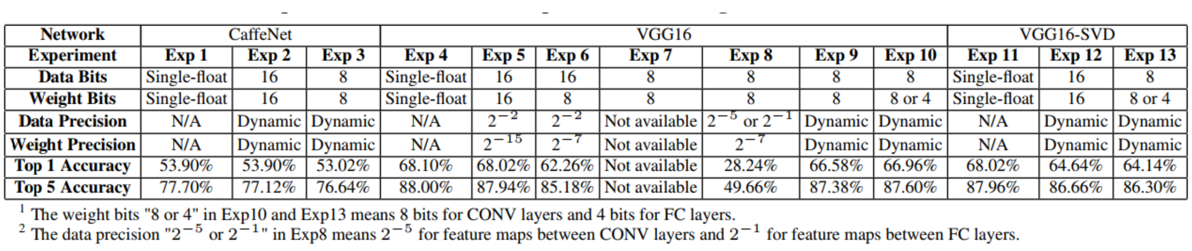
Qiu et al proposed to dynamically change the digits where the decimal point is placed for each layer in order to prevent decreasing accuracy [3]. The fixed-point representation can be expressed by the following equation.

$$n = \sum_{i=0}^{bw-1} B_i \cdot 2^{-f_l} \cdot 2^i \tag{2.4}$$

bw is a bit-width and f_l is a digit to which the decimal point is placed. For the value of each layer, the appropriate f_l when given bw are decided depending on the following formula.

$$f_l = \underset{f_l}{\operatorname{argmin}} \sum |W_{float} - W(bw, f_l)| \tag{2.5}$$

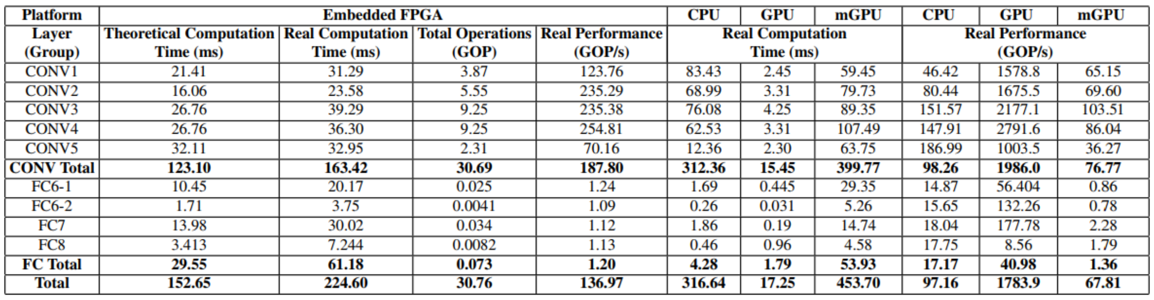
As a result, it is possible to decide the position of the decimal point so that the error becomes the smallest as compared with the floating point.



Network	CaffeNet			VGG16							VGG16-SVD		
Experiment	**Exp 1**	**Exp 2**	**Exp 3**	**Exp 4**	**Exp 5**	**Exp 6**	**Exp 7**	**Exp 8**	**Exp 9**	**Exp 10**	**Exp 11**	**Exp 12**	**Exp 13**
Data Bits	Single-float	16	8	Single-float	16	16	8	8	8	8	Single-float	16	8
Weight Bits	Single-float	16	8	Single-float	16	8	8	8	8	8 or 4	Single-float	16	8 or 4
Data Precision	N/A	Dynamic	Dynamic	N/A	2^{-2}	2^{-2}	Not available	2^{-5} or 2^{-1}	Dynamic	Dynamic	N/A	Dynamic	Dynamic
Weight Precision	N/A	Dynamic	Dynamic	N/A	2^{-15}	2^{-7}	Not available	2^{-7}	Dynamic	Dynamic	N/A	Dynamic	Dynamic
Top 1 Accuracy	53.90%	53.90%	53.02%	68.10%	68.02%	62.26%	Not available	28.24%	66.58%	66.96%	68.02%	64.64%	64.14%
Top 5 Accuracy	77.70%	77.12%	76.64%	88.00%	87.94%	85.18%	Not available	49.66%	87.38%	87.60%	87.96%	86.66%	86.30%

[1] The weight bits "8 or 4" in Exp10 and Exp13 means 8 bits for CONV layers and 4 bits for FC layers.
[2] The data precision "2^{-5} or 2^{-1}" in Exp8 means 2^{-5} for feature maps between CONV layers and 2^{-1} for feature maps between FC layers.

Figure.2.7: Exploration of different data quantization strategies with state-of-the-art CNNs [3].



Platform	**Embedded FPGA**				**CPU**	**GPU**	**mGPU**	**CPU**	**GPU**	**mGPU**
Layer (Group)	**Theoretical Computation Time (ms)**	**Real Computation Time (ms)**	**Total Operations (GOP)**	**Real Performance (GOP/s)**	**Real Computation Time (ms)**			**Real Performance (GOP/s)**		
CONV1	21.41	31.29	3.87	123.76	83.43	2.45	59.45	46.42	1578.8	65.15
CONV2	16.06	23.58	5.55	235.29	68.99	3.31	79.73	80.44	1675.5	69.60
CONV3	26.76	39.29	9.25	235.38	76.08	4.25	89.35	151.57	2177.1	103.51
CONV4	26.76	36.30	9.25	254.81	62.53	3.31	107.49	147.91	2791.6	86.04
CONV5	32.11	32.95	2.31	70.16	12.36	2.30	63.75	186.99	1003.5	36.27
CONV Total	**123.10**	**163.42**	**30.69**	**187.80**	**312.36**	**15.45**	**399.77**	**98.26**	**1986.0**	**76.77**
FC6-1	10.45	20.17	0.025	1.24	1.69	0.445	29.35	14.87	56.404	0.86
FC6-2	1.71	3.75	0.0041	1.09	0.26	0.031	5.26	15.65	132.26	0.78
FC7	13.98	30.02	0.034	1.12	1.86	0.19	14.74	18.04	177.78	2.28
FC8	3.413	7.244	0.0082	1.13	0.46	0.96	4.58	17.75	8.56	1.79
FC Total	**29.55**	**61.18**	**0.073**	**1.20**	**4.28**	**1.79**	**53.93**	**17.17**	**40.98**	**1.36**
Total	**152.65**	**224.60**	**30.76**	**136.97**	**316.64**	**17.25**	**453.70**	**97.16**	**1783.9**	**67.81**

Figure.2.8: Performance of different platforms with VGG16-SVD network [3].

Figure.2.7 shows the result of comparing the accuracy when changing the expression of decimal numbers in some CNN models. It is found that accuracy of fixed-point 16bit does not decrease compared with the accuracy of floating-point 32bit. However, in the case of fixed-point 8 bits, inference does not work because both the convolutional layer and the fully connected layer can not be expressed by one type of the position of decimal point(Figure.2.7 Exp7). Inference works by adopting different decimal point positions in the convolutional layer and the fully connected layer, but the accuracy is lowered by nearly 40% as compared with the case of floating-point 32bit(Figure.2.7 Exp8). By using the method of Qiu et al., if the optimal decimal point positions are used for each layer, it is possible to do inference with fixed-point 8 bits without decreasing accuracy(Figure.2.7 Exp9).

Figure.2.8 shows the result of comparing FPGA implementation when optimizing memory access by rearranging data in the order of access in addition to optimizing the position of the decimal point with implementations of Intel Xeon E5-2690 CPU@2.90GHz, Nvidia K40 GPU (2880 CUDA cores with 12GB GDDR5 284-bit memory) and Nvidia TK1 Mobile GPU development kit (192 CUDA cores).

As a result, the computing performance of convolutional layers is 1.4 times and 2.0 times better compared with CPU ·mGPU, respectively. Compared with GPU, GPU has 13.0 times higher computing performance, but GPU consumed 26.0 times power. As for the fully connected layers, the performance of CPU was 17.17GOP/s and the performance of GPU was 40.98GOP/s, whereas in the case of FPGA, it was 1.20GOP/s. This is because the ratio of data transfer is large in the fully connected layers, and the calculation performance is restricted by the bandwidth. It is considered that the performance of mGPU is low for the same reason.

Although it still has some difficulties, especially for fully connected layers, reducing the bit width using the fixed point is effective in increasing the efficiency of the calculation.

So far I have introduced research to accelerate the computation of machine learning using FPGAs. In order to efficiently machine learning on the FPGA, I found that it is important to reduce the amount of data transfer by loop tiling or reducing bit width. Reducing bit width not only reduces the amount of data transfer but also has a possibility of simplifying the calculation. So, I state researches of compressing CNN models including reduction of bit width.

2.3 Model Compression

2.3.1 Quantized Convolutional Neural Networks

There are several studies to reduce the bit width of the weight during training [20, 21, 22]. These studies use MNIST as data sets, and a multilayer perceptron consisting only of fully connected layers. The techniques of reducing the bit width of CNN weights for tasks of 10 class classifications such as MNIST and CIFAR-10 are also studied [9, 23, 8]. These techniques succeeded in binarizing weights and activation without degrading the inference accuracy. Several researches showed studies reducing the bit width of the weight and activation in CNN for large-scale data sets such as ImageNet [24, 10, 13]. One key difference with these studies and our current study is that they needed to train models with binarized weights and activation.

On the other hands, [14, 13, 15] are aim to reduce bit width for already-trained models similar to our study. Lin et al compress the model by optimizing the bit width of each layer so as to minimize the fixed point quantization error [14]. Their approach only applies to the convolution layer, so it cannot compress well an ImageNet model, which the fully connected layers dominate the model size, whereas it can compress a CIFAR-10 model, which the convolutional layers dominate, by $> 20\%$. Kamiya et al decomposed real-valued vector of weights to binary basis vectors and quantized activation [15]. As the result, real-valued inner-product computation is replaced with binary inner-product computation and they succeeded to accelerate the computation of inference and decrease model size. Miyashita et al showed that it is possible to reduce the weight of the fully connected layers to 4 bits and the weight of the convolutional layers to 5 bits without the large decreasing of accuracy, by expressing the values using logarithms based on 2 or $\sqrt{2}$[13].

2.3.2 Adaptive Quantization

We introduce Variable Bin-size Quantization (VBQ), which is a quantization using different bin sizes for each quantization bin. VBQ is a type of adaptive quantization. In signal compression, adaptive quantization schemes is widely used, in which an appropriate bin size are chosen for each sample to be quantized. This idea was applied to CNN in [25, 26, 27], though the numerical expression was adapted within the fixed point scheme and bin sizes were not variable.

2.3.3 Other Approaches

Other approaches to handle the complexity of large CNN models include (a)training a small model with the output of a large and high accuracy model [28, 29] (b)sharing weights using feature hash [30] (c)combining pruning and quantization during training [31]. These approaches are complementary to our proposed approach and it could be possible to compress the model more by using those together.

Chapter 3

Investigation of Weights in CNN

3.1 Framework For Our Experimen

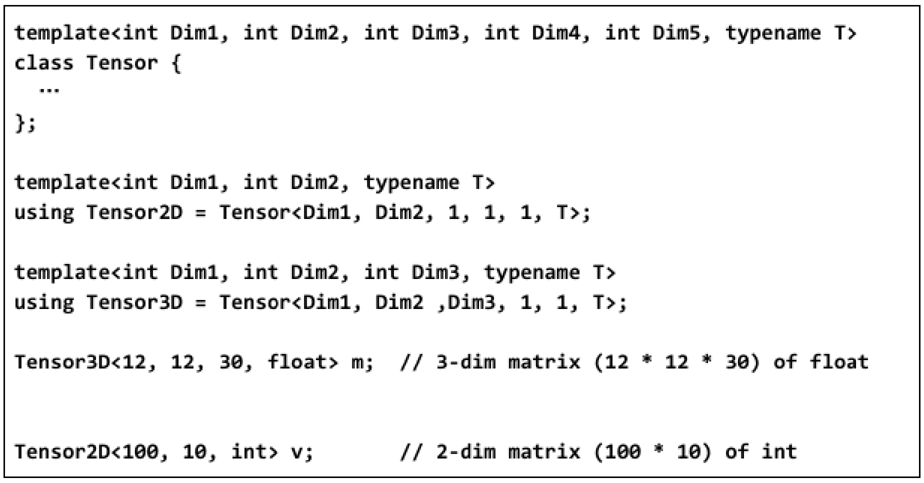
We first investigated the properties of the trained weights before studying the method of quantizing the trained weights. I have two purposes for this investigation. The first is to confirm whether FP32 which is generally used is really sufficient as a baseline. The second is to understand the distribution of the trained weights to be used for reference when considering the quantization method. For this investigation, I need a framework that can easily change the numerical expression in CNN calculation.

3.1.1 Detail of Implementation

I implemented a framework that allows CNN computation with various numerical expressions in C++. The calculation of CNN and other neural networks is primarily a matrix operation. In general, matrices are implemented using multidimensional arrays. In the calculation of CNN, fixed length array is appropriate because the number of elements of each matrix never change in the middle. On the other hand, in the calculation process of CNN, there are many situations in which similar calculations are performed for arrays of various sizes. For this reason, it is necessary to implement each function so that it does not depend on the array size. So, I implemented it using "template" as shown in Fig3.1. Dim-n represent the size of the n dimension. Only 4-dimensional matrix at most appears in the calculation of CNN. So I implemented it to handle up to 5 dimensions with enough margin. I also implemented each function using "template" in the same way so that any type of matrix can be handled uniformly.

3.1.2 Unit Testing

It is important to make sure that the created program is working properly. It is called unit testing to divide a program by element and test whether each element is working correctly. I conducted a unit testing to verify that the CNN framework I created works properly. In



```
template<int Dim1, int Dim2, int Dim3, int Dim4, int Dim5, typename T>
class Tensor {
  …
};

template<int Dim1, int Dim2, typename T>
using Tensor2D = Tensor<Dim1, Dim2, 1, 1, 1, T>;

template<int Dim1, int Dim2, int Dim3, typename T>
using Tensor3D = Tensor<Dim1, Dim2 ,Dim3, 1, 1, T>;

Tensor3D<12, 12, 30, float> m;  // 3-dim matrix (12 * 12 * 30) of float


Tensor2D<100, 10, int> v;       // 2-dim matrix (100 * 10) of int
```

Figure.3.1: Tensor class

order to do unit testing, it is necessary to prepare correct answers. For this time, I used the calculation results of each function of existing Python implementations*1*2 as correct answers.

3.1.3 Numerical Expression

Various numerical expressions can be handled in my framework. Even if it is not a type supported by C ++ such as int or float, it can be run by defining the following.

- comparison operator (==, >, <, >=, <=)
- assignment operator (=)
- arithmetic operator (+, *, -, /)
- cast operator to/from float

Since it is difficult to define random function and exponential function to own type, I implemented it by using those function for costing from float.

3.2 Experimental results and Analysis

I trained CNN using various numerical expressions. I investigated the distribution of weights and inference accuracy when training converged. In order to see only the influence due to the difference in numerical expression, I used the same seed for pseudo random function. I used SimpleConvNet in Figure.3.2, which is composed a convolutional layer and two fully connected layers and floating point 16bit(FP16), 32bit(FP32), 64(FP64) as numerical expressions.

*1 https://github.com/oreilly-japan/deep-learning-from-scratch
*2 http://www.numpy.org/

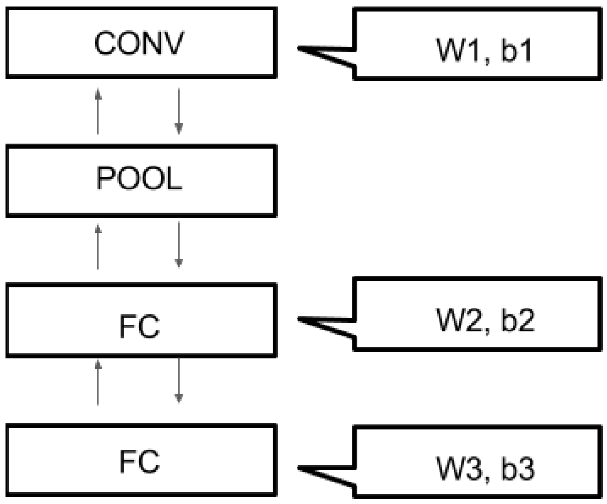


Figure.3.2: SimpleConvNet

Table.3.1: Accuracy of Each Numerical Expression

FP16	FP32	FP64
0.9347	0.9782	0.9784

3.2.1 Accuracy

The accuracy was almost the same for FP32 and FP64, about 97.8%, and in FP16 it was 93.5 % which is 4% lower than those of FP32 and FP64 in Table.3.1.

3.2.2 Histograms

I created and compared histogram for each weights. The case of W1 is shown in Figure.3.3. The upper left is the initial value of W 1, the upper right is W 1 trained by FP 16, the lower left is W 1 trained with FP 32, and the lower right is W 1 histogram trained with FP 64. I found the histograms are very similar between FP32 and FP64. On the other hand, when comparing them with FP16, it can be seen that around 0 has large population in FP16. Also, the range of values is approximately $[-0.75, 0.75]$ for FP32 and FP64, while it is narrow for FP16 as being approximately $[-0.25, 0.25]$. Since the range of the initial value is also approximately $[-0.25, 0.25]$ as in FP16, the range expanded as training progressed in FP32 and FP64, but it did not spread in FP16. A similar tendency was also observed for parameters other than W1. The features found from the histogram are summarized as follows.

- Distributions of FP32 and FP64 are very similar
- In FP16, there are more values of 0 than FP32 and FP64
- In FP32 and FP64, the range widened to about three times the range of the initial value

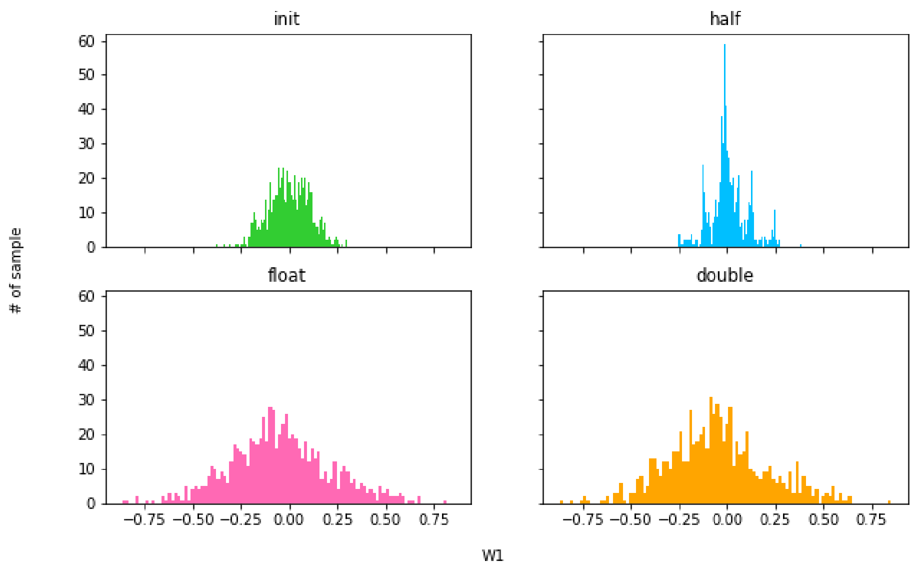


Figure.3.3: Histogram of W1

Table.3.2: Cross Correlation

	FP16 - FP64	FP32 - FP64	FP64 - FP16
W1	0.655	0.963	0.661
W2	0.515	0.994	0.516
W3	0.490	0.966	0.500

- The range of FP16 does not differ from the range of the initial value

Moreover, in order to quantitatively evaluate how similar the value of the parameter is in each numerical expression, I showed the cross correlation coefficient of each. The results are shown in Table.3.2. This result also shows that there is a strong correlation between the parameters trained in FP32 and FP64.

3.2.3 Relationship between Numerical Expression and Accuracy

As shown in Table.3.1, it was found that the accuracy of FP16 is 4% lower than that of FP32 and FP64. There are two possible reasons for this. The first is that FP16 is not enough for training. The second is that FP16 is not enough for inference. In order to ascertain which is the cause, I compared parameters trained in FP64 and cast to FP16 with parameters trained in FP16. Also for FP32, I compered parameters trained in FP64 and cast to FP32 with parameters trained in FP32. I showed the result in Table.3.3. It was found that the almost same accuracy as FP64 can be obtained even if the inference is performed in FP16

Table.3.3: Comparison of Accuracy of FP64 Training with FP16/FP32 Training

	Train FP64	original
FP16	0.9782	0.93347
FP32	0.9784	0.9782

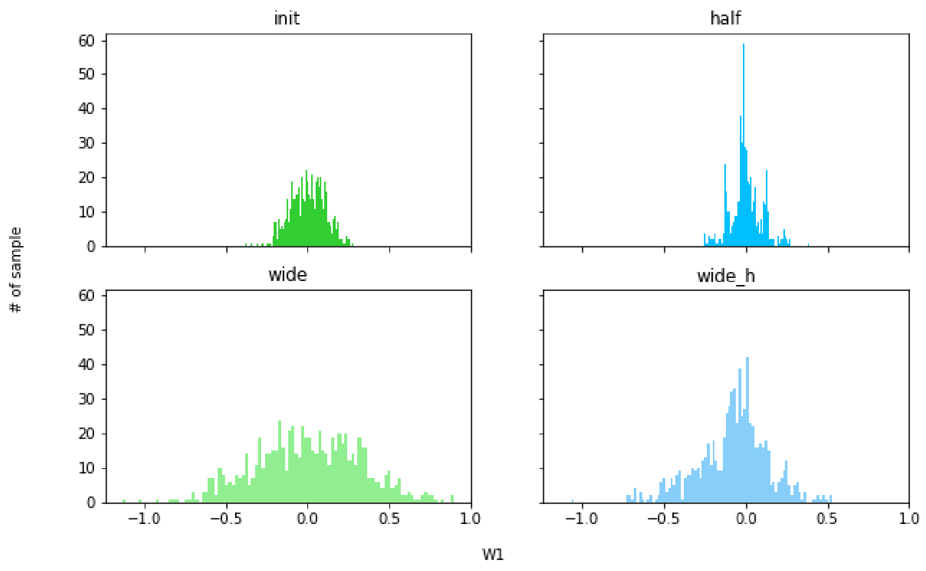


Figure.3.4: Histogram of W1 ($\sigma = 0.3$)

and FP32 using the parameters trained in FP64. Even if casting the parameter trained in FP64 to FP16, the accuracy did not decrease much. It is possible to express parameters that can obtain accuracy that is almost the same as 64 bits by at least 16 bits. In addition, it has become clear that the reason why the accuracy decreases when training is performed in FP16 is that the parameters are not sufficiently optimized as compared with FP64.

3.2.4 Relationship between Value Range and Accuracy

As shown in Figure.3.3, it was found that when training is performed with FP 16, the range of the parameter hardly spreads out from the range of the initial value. In the experiment of Figure.3.3, I used the normal distribution with average 0 and standard deviation 0.1 as the initial value. The range of parameter in FP16 was about $[-0.25, 0.25]$. On the other hand, the range of parameter trained in FP32 and FP64, which gave 4% or more accuracy than that trained in FP 16, was $[-0.75, 0.75]$. I considered the range of parameter in FP16 was supposed to widen $[-0.75, 0.75]$ by using initial value of the normal distribution with average 0 and standard deviation 0.3. I also expected to improve the accuracy by doing so. I showed the result in Figure.3.4 When tripling the standard deviation of the distribution of the initial values, the range of parameters trained in FP16 also triples. The range of parameters was almost the same as the value range of parameters trained in FP64 which gave about 98% accuracy. However, the accuracy was 93.1 %, which was1% lower than before expanding the range. It is found that the range of parameters has no relation to the decrease in the accuracy and there are other reasons of decreasing accuracy when training is conducted in FP16.

Chapter 4

Optimizing Quantization

4.1 Variable Bin-size Quantization

When converting a continuous value to a digital value, quantization is used. Quantization is an operation that converts a range of values to a representative value. I denote the range by bin, the width of the range by bin size, and the boundary of the range by bin boundary. The number of bins corresponds to 2^n, where n is the bit width of the numerical expression used in the quantization.

In order to optimize the quantization bins more freely, I propose "Variable Bin-size Quantization (VBQ)" as shown in Figure 4.1. VBQ is a type of adaptive quantization where representative values can be changed independently, so as to optimize bin locations and bin sizes. Here, the bin boundary is taken at the middle of two adjacent representative values.

4.2 Genetic Algorithm

A genetic algorithm is used to optimize representative values v_i of VBQ by taking the accuracy of CNN to be the fitness score. Genetic algorithms are heuristic algorithms that search for an optimal solution by repeating operations such as reproduction, crossover, and mutation on multiple individuals with various genomes. In this study, each "genome" is a set of representative values $\{v_i\}$, and is a candidate of an optimized numerical representation for CNN. In the following, procedures of the employed algorithm are outlined.

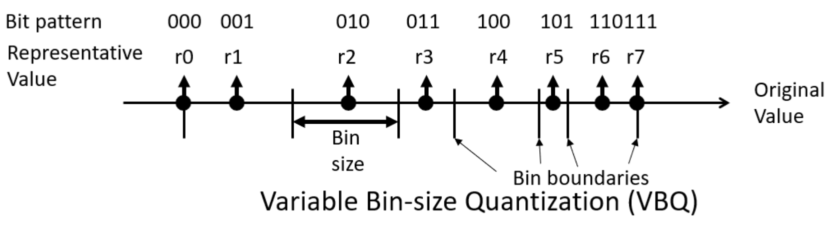


Figure.4.1: Variable Bin-size Quantization example

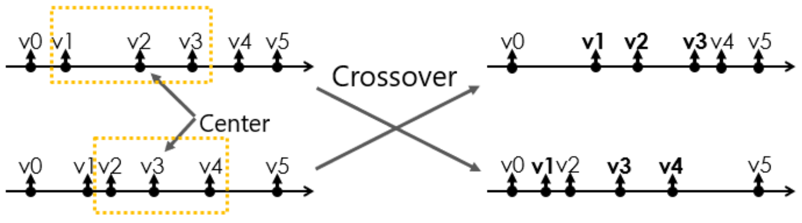


Figure.4.2: Diagram illustrating the modified crossover method implemented in the genetic algorithm.

4.2.1 Reproduction

Reproduction is an operation to succeeds superior genomes to the next generation. In the present algorithm, I combine the elite selection and the roulette selection as follows. First, I select an individual with the highest fitness, and reproduce it in the next generation. From the rest of individuals, R_r% are selected according to the probability proportional to the fitness.

4.2.2 Crossover

Crossover (also known as recombination) is an operation that combines information from two individuals to generate offspring by mating. There are various methods for the crossover operation, but the two-point crossover is widely used, where two crossover points are randomly chosen and a section of genomes between them are exchanged. In the present algorithm, I choose two sets of R_c% of individuals randomly, and apply a variation of the two-point crossover operation on each pair of them as follows.

In an ordinary two-point crossover, indexes of the genetic sequence are taken as the crossover points. However, in the VBQ optimization, where the genetic sequence is an array of representative values, the range between two crossover indexes $[v_i, v_j]$ may not overlap at all between two individuals. Therefore, instead of specifying two crossover indexes, I specify a center of the range and the number of representative values to be exchanged around the center. In this manner, genetically comparable information can be exchanged as shown in Figure 4.2.

4.2.3 Mutation

Mutation is an operation to randomly change a part of a gene and serves to promote genetic diversity. This then avoids local optimal solutions by preventing the genes from becoming too similar. I implemented mutation by randomly moving the position of the randomly selected representative values. The amount of the movement is regulated such that the value does not exceed the representative values on both sides. I choose R_m% of individuals randomly, and

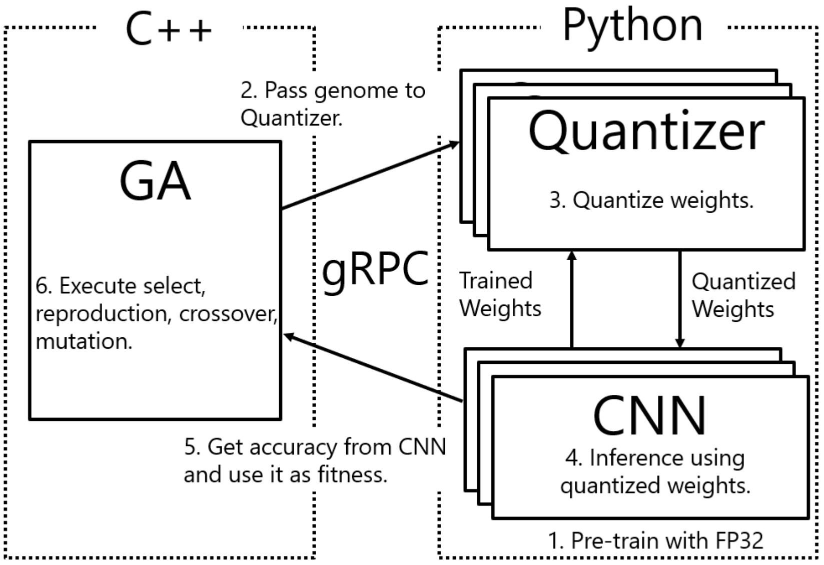

C++
Python
2. Pass genome to Quantizer.
Quantizer
3. Quantize weights.
GA
6. Execute select, reproduction, crossover, mutation.
gRPC
Trained Weights
Quantized Weights
CNN
4. Inference using quantized weights.
5. Get accuracy from CNN and use it as fitness.
1. Pre-train with FP32

Figure.4.3: Diagram showing the overall steps in the experimental system.

apply the mutation above.

4.3 Outline of Experiments

4.3.1 Outline

The outline of the system used in this experiment is shown in Figure 4.3. The system consists of three parts: GA, Quantizer, and CNN. The flow of the experiment is as follows.

1. CNN: Train with FP32, and create trained weights.
2. GA: Give genomes to Quantizer to evaluate genomes fitness.
3. Quantizer: Quantize the trained weight of CNN with the received genomes as the bin boundary.
4. CNN: Inference using quantized weights, and pass the obtained the Top-1 accuracy to GA.
5. GA: Use the Top-1 accuracy as the fitness of that genomes.
6. GA: Execute the reproduction, crossover and mutation described in Section 4.2 to create the next generation genomes.
7. Repeat 1 ~ 6

We experimented with VGG16, which is composed 13 convolutional layers and 3 fully connected layers as shown in Table.4.1, using the ILSVRC2012 validation dataset. Initial genomes were randomly generated, and the number of genomes was 50. The probability of reproduction (R_r), crossover ($2R_c$), and mutation (R_m) were 30%, 50%, and 20%.

Layer#	Layer	Param #
Layer1	Conv2D	$3 \times 3 \times 64$
Layer2	Conv2D	$3 \times 3 \times 64$
	MaxPooling	
Layer3	Conv2D	$3 \times 3 \times 128$
Layer4	Conv2D	$3 \times 3 \times 128$
	MaxPooling	
Layer5	Conv2D	$3 \times 3 \times 256$
Layer6	Conv2D	$3 \times 3 \times 256$
Layer7	Conv2D	$3 \times 3 \times 256$
	MaxPooling	
Layer8	Conv2D	$3 \times 3 \times 512$
Layer9	Conv2D	$3 \times 3 \times 512$
Layer10	Conv2D	$3 \times 3 \times 512$
	MaxPooling	
Layer11	Conv2D	$3 \times 3 \times 512$
Layer12	Conv2D	$3 \times 3 \times 512$
Layer13	Conv2D	$3 \times 3 \times 512$
	MaxPooling	
Layer14	FC	25088×4096
Layer15	FC	4096×4096
Layer16	FC	4096×1000

Table.4.1: VGG16

4.3.2 System Implementation

The GA was implemented in C++, and the Quantizer and CNN were implemented in Python. I used Keras*1 and Tensorflow*2 for CNN implementation. Separating the system to three parts: GA, Quantizer, CNN allows us to change the optimization method to other than GA or to fiddle with the CNN model without effect to other parts. Thanks to that, it is easy to update or maintain the program. At first, the GA part and the Quantizer and CNN part were each one process. However, there was a problem that calculation became very time consuming when the model of CNN became large. The reason is because for calculating one generation of GA, the calculation of Quantizer and CNN are repeated the same times

*1 https://keras.io/

*2 https://www.tensorflow.org/

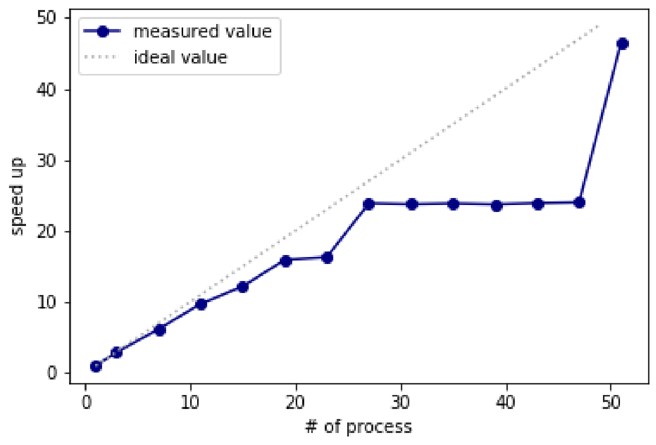


Figure.4.4: The result of execution time of GA with 50 genomes.

as the number of genes. Also, since Reedbush and ABCI, which I used, have more than one GPU on one node, performing CNN in one process can only use one GPU and it is inefficient. Therefore, I improved the system so that Quantizer and CNN parts can be performed on multiple processes for the one GA process.

Originally I used gRPC[*3] to communicate between the GA part and the Quantizer and CNN part. gRPC communicates using Protocol Buffers[*4], which is a binary format containing type information. gRPC can communicate without any problems between different programming languages since Protocol Buffers does not depend on programming languages. It is theoretically possible to perform multi-node communication using gRPC on a spar computer, but it is a little troublesome, because some settings, like the path of the file of the IP address or host name that can be used, the usable port etc..., are different depending on the supercomputer.

MPI is generally used for inter-node communication of supercomputers. MPI makes inter-process communication easy. Also, the one program can be executed without being conscious of the difference for each supercomputer because MPI is set up for each supercomputer in advance. However, communication between programs written in different programming languages is not assumed. Because my system is written in a different language in GA part and Quantizer and CNN part, it is difficult to communicate with MPI.

For the reasons, I used gRPC and MPI together. I implemented a part called ”Communicator” that relays MPI and gRPC in C ++. By multi-processing including Communicator, I can use MPI for inter-process communication and gRPC for conversion from C ++ to Python.

The problem of communication has been solved so far. Next, I solved a problem of collision of

[*3] https://grpc.io/
[*4] https://developers.google.com/protocol-buffers/

Layer#	Layer	Param #
Layer1	Conv2D	$3 \times 3 \times 64$
Layer2	Conv2D	$3 \times 3 \times 64$
	MaxPooling	
Layer3	Conv2D	$3 \times 3 \times 128$
Layer4	Conv2D	$3 \times 3 \times 128$
	MaxPooling	
Layer5	Conv2D	$3 \times 3 \times 256$
Layer6	Conv2D	$3 \times 3 \times 256$
Layer7	Conv2D	$3 \times 3 \times 256$
	MaxPooling	
Layer8	FC	200704×1024
Layer9	FC	1024×512
Layer10	FC	512×10

Table.4.2: VGG-like model

resources when running multiple processes in one node. I implemented CNN using tensorflow, but tensorflow reserves all GPU's memory. For example, when there are 4 GPUs in one node, if you launch multiple CNN processes on one node, the first process reserves all memory of the 4 GPUs even though only 1 GPU is used for calculation. Other processes abend due to insufficient memory. For resolving it, I implemented to specified the GPU to use by rewriting the environment variable when launching the CNN process, and make the other GPUs invisible.

The results of GA with 50 genomes are shown in Figure.4.4. The x axis represents the number of processes and the y axis represents the ratio with the execution time in the case of one process. The reason that the graph is in a staircase is because the execution time of a process with a large number of calculations becomes a bottleneck. Figure.4.4 shows that multiple processing and parallelization are done correctly.

4.4 Preliminary experiment

I carried out a preliminary experiment using a small VGG-like model with CIFAR-10 dataset, shown in Table4.2, before starting full-scale experiment. The purpose of this experiment is the following two.

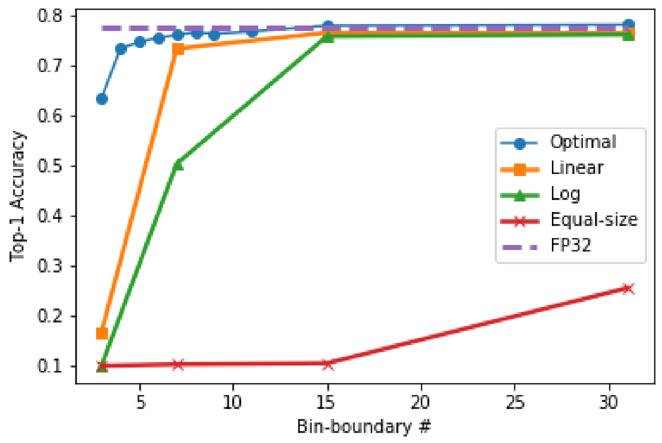


Figure.4.5: Relationship between the number of bin boundaries and Top-1 Accuracy in some quantization methods.

- To make sure that my system is working properly
- To ascertain whether using optimized quantization is really more suitable for reducing bit width in CNN than using artificial quantization like equidistance or logarithmic spacing

I have optimized VBQ by varying the quantizing methods and the number of bin boundaries from 4 to 31. The resulting best Top-1 accuracy is plotted in Figure. 4.5. It was confirmed that the system is working correctly from the fact that comparable accuracy can be obtained before quantization. Three artificial bin boundary distributions are investigated. In "Equal-size" distribution, bins are arranged such that each bin contains the same number of trained weights. In "Linear" distribution, bin sizes are taken to be the same. And in "Log" distribution, bin sizes are increased exponentially as the bin location departs from zero. Roughly speaking, the Linear and Log distributions correspond to the fixed point and the floating point numerical expressions, respectively. For these distribution, I took 3σ range to be the full dynamic range, where σ is the standard deviation of the trained weights. The Top-1 accuracy obtained from them are plotted in Figure. 4.5. It was found that the accuracy when using the optimized bin boundary becomes higher than the other bin boundaries despite the number of bins. I demonstrated that the system is working correctly and the effectiveness of optimization. So, I start full-scale experiment in the next section.

4.5 Experimental Results and Discussion

We experimented in the cases where the representative value was positive/negative symmetric and asymmetric with VGG 16. I used the trained weights available from Keras library. I obtained 67.38 % accuracy with FP32 in the validation data set of ILSVRC2012.

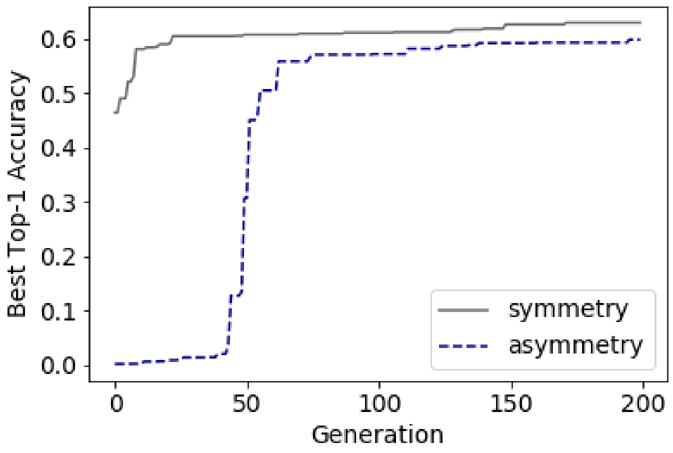


Figure.4.6: Changes in the accuracy for each generation

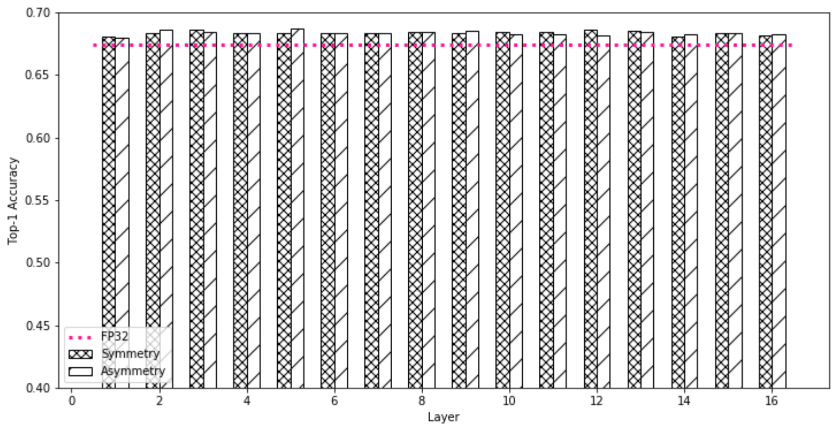


Figure.4.7: The accuracy of quantizing and optimizing only one layer with other layer using FP32

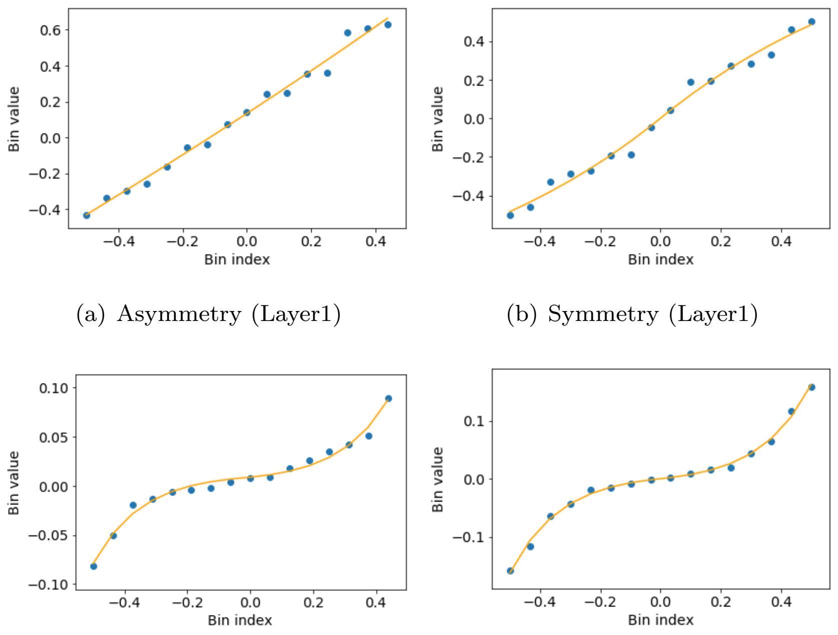
The number of representative values to be optimized was 16. Changes in the accuracy for each generation with the same quantization applied to all layers are shown in Figure.4.6.

We can get to a better solution quickly when the representative values are positive/negative symmetric. Also, I showed the result of quantizing and optimizing only one layer with other layer using FP32 in figure 4.7. I found quantization which can exceed the base accuracy with FP32 in all layers regardless of symmetry.

Next, I conducted nonlinear regression analysis in order to investigate the distribution of the optimized representative values. I heuristically deduced Eq.4.1 as the regression equation.

$$V(x) = sign(x - d) \times b \times (a^{|x-d|} - 1) + c \tag{4.1}$$

In Eq.4.1, a is an exponential index, b is a scaling factor, c is a shift in the y-direction, and d is a bias in the x-direction. Eq.4.1 is a exponential function when a is a large value and it can be approximated as a linear function when a is small. Now $V(x)$ is the bin boundary place and



(a) Asymmetry (Layer1) (b) Symmetry (Layer1)

(c) Asymmetry (Layer10) (d) Symmetry (Layer10)

Figure.4.8: Distributions of representative values and the regression results

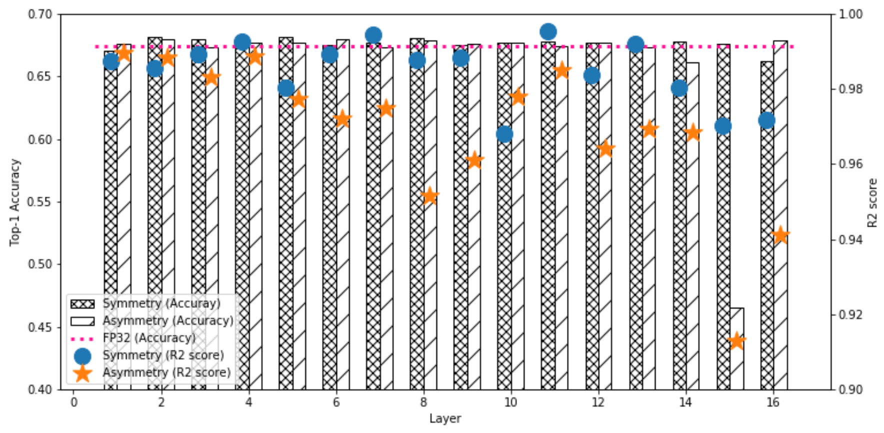


Figure.4.9: The accuracy of quantizing each layer using regression results

x is the index of the representative values, normalized with $[-0.5, 0.5]$. The regression results for the first and tenth layers are shown in Figure. 4.8 for the case of symmetry and the case of asymmetry.

In both symmetry and asymmetry cases, the first layer is nearly linearly distributed and the tenth layer is close to the exponential distribution, and I could fit successfully in both cases. The accuracy when choosing a representative value along the fitting function for each layer is shown in Figure.4.9. In the case of symmetry, there is no layer much lower than the accuracy of FP32. In the case of asymmetry, on the other hand, the accuracy remarkably decreased at the 14th layer. Considering that the R2 score of the 14th layer is lower than that of the other layers, the reason is that regression didn't work well. For the other layers also the symmetric case has a higher R2 score than asymmetry case. I found that the regression goes well in the symmetry case. In addition, when the representative value is taken symmetrically, since c, d in Eq.4.1 is theoretically 0, Eq.4.1 becomes Eq.4.2, which has an advantage of being simple.

$$F(x) = sign(x) \times b \times (a^{|x|} - 1) \tag{4.2}$$

Therefore, I decided to experiment only in the case of placing representative values in positive/negative symmetry hereafter.

In the case of optimizing the representative values itself, the accuracy exceeds that of FP32 int all layers. On the other hand, in the case of taking the representative values among the regression function, the accuracy has fallen below that of FP32 in some layers. However, in the case of symmetry, the decrease is only about 1.2%. Besides, when represent values are placed along Eq.4.2, It is enough to optimize two values(a, b) for each layer, regardless of the number of representative values. So, the number of parameters to optimize can be greatly reduced especially when the number of representative values is large. Therefore, instead of optimizing the representative values itself, it is useful to optimize a, b and take representative values along the Eq.4.2. In the next chapter, I will consider how to optimize a, b directly without regression.

Chapter 5

Optimizing Fitting Parameters

5.1 Grid Search

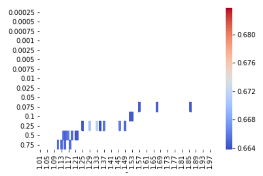
First, I draw heatmaps of the Top-1 accuracy against parameters (a, b), to catch what the search space looks like. For better survey, I employ $\tilde{a} = a^{1/N}$ instead of a in constructing the uniform grid, where N is the number of bins. Logarithmic grid is also employed for b. The results are summarized in Fig. 5.1. The accuracy at FP32 is taken as the baseline, and those points deviated more than 1 % from the baseline are omitted from the plots. The red and blue regions in the plots correspond above and below the baseline, respectively.

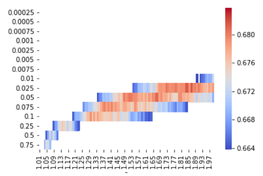
As shown in the figure, the shape of the heatmap is similar among layers, though the the size of the high accuracy region differ. Especially, the high accuracy regions are wide for the fully connected layers (Layer 13–15), indicating that the accuracy is insensitive to the quantization parameters. For these layers, it may be possible to reduce the quantization bit width further. On the contrary, the convolutional layers are sensitive to the quantization parameters, especially for the Layer 1. It is probably because the weights are heavily reused in the convolutional layer, which causes a slight quantization error to accumulate. These results indicate that the bit width should also be optimized from layer to layer, though it is beyond the scope of this paper.

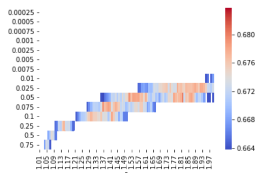
As a result of heatmap, it was found that the accuracy changed smoothly with respect to a and b. Therefore, it is possible to get the optimum solution of a and b by updating a and b in the direction that the accuracy becomes high. Meanwhile, I also found that there are multiple spots where the accuracy is higher than that around them. Therefore, updating a and b only in a direction in which accuracy gets higher may possibly lead to a local maximum. From the above, I decided to optimize a and b by Simulated Annealing.

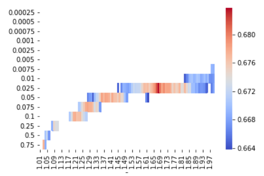
5.2 Simulated Annealing

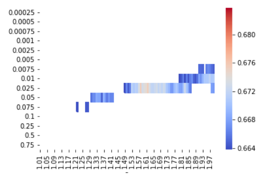
In the heatmap drawn in the previous section, I notice a lot of local maximums. To optimize the parameters a and b avoiding those local maximums, I employ the simulated annealing (SA)

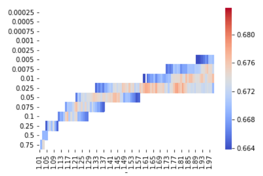

(a) Layer1

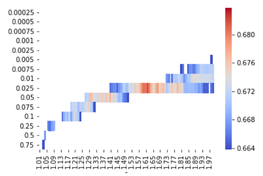

(b) Layer2

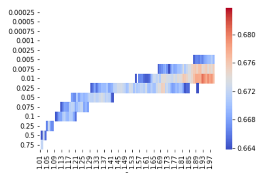

(c) Layer3

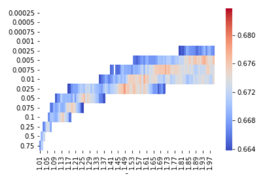

(d) Layer4

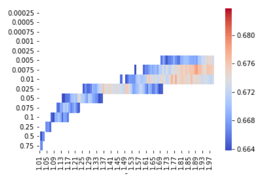

(e) Layer5

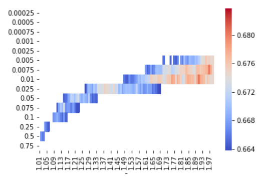

(f) Layer6

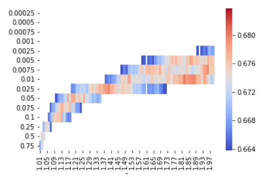

(g) Layer7

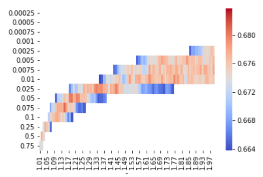

(h) Layer8

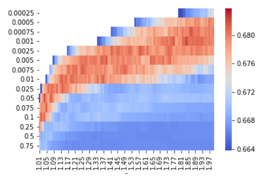

(i) Layer9

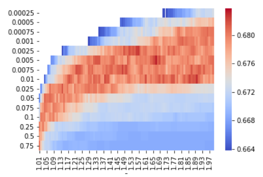

(j) Layer10


(k) layer11


(l) Layer12


(m) Layer13


(n) Layer14


(o) Layer15

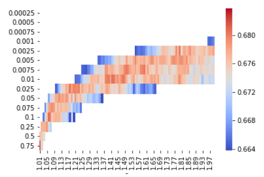

(p) Layer16

Figure.5.1: Heatmap of a, b in 8 represent values(3 bit)

algorithm.

SA is an optimization algorithm that simulates annealing in metal engineering [32, 33]. SA utilizes an artificial thermal fluctuation to prevent from sticking at the local optimum solutions. Setup of the SA simulation is simple: just replace the GA portion in Fig.4.3 by SA. One iteration of our algorithm goes as follows.

1. SA: selects neighborhood a , b of a, b
2. SA: generates a genome with $v_i = V(x_i|a\ , b\ )$ and sends it to Quantizer
3. Quantizer and CNN: perform inference to obtain the Top-1 accuracy e
4. SA: updates a, b by a , b with the probability $P(e, e\ , T)$
5. Repeat 1 ~ 6 for each layer
6. SA: lowers the tempreture T

In step 1, I prepare Δa and Δb randomly in the range of $[-\frac{1}{2}aT, \frac{1}{2}aT]$ and $[-\frac{1}{2}bT, \frac{1}{2}bT]$,

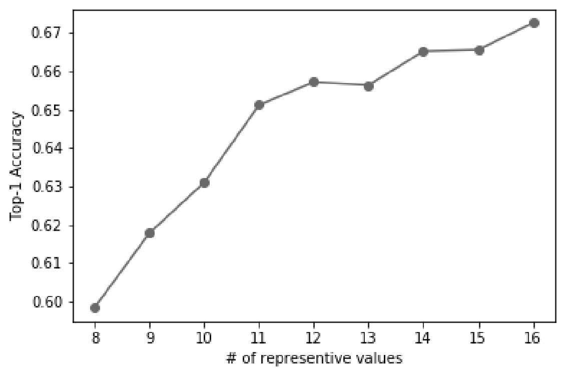


Figure.5.2: Changes in the accuracy for the number of representative values

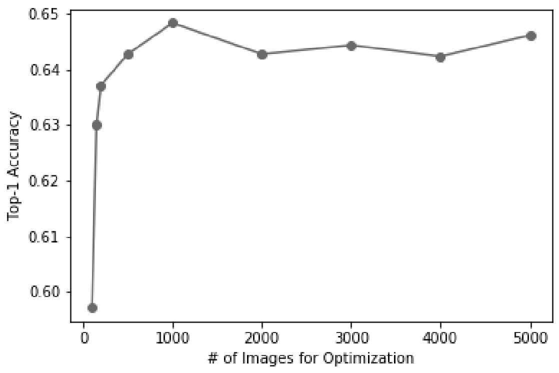


Figure.5.3: The number of images for Optimization (16 representative values)

respectively, and generate three neighborhoods, $(a+\Delta a, b)$, $(a, b+\Delta b)$, and $(a+\Delta a, b+\Delta b)$. In step4, probability $P(e, e\ , T)$ is defined as

$$P(e, e\prime, T) = \begin{cases} 1 & (e\prime \geq e) \\ \exp\left(\frac{(e\prime - e)\times 100}{T}\right) & (e\prime < e) \end{cases} \tag{5.1}$$

where e and e are the Top-1 accuracies calculated with (a, b) and $(a\ , b\ )$, respectively. The initial values of (a, b) are taken to be $(1.25, 0.05)$ for all layers, based on the heatmap, and the initial temperature is taken at $T = 1$ with the temperature reduction rate of 0.95. The simulation is stopped when (a, b) of all layers are not updated for consecutive 30 iterations.The SA simulations are performed by varying the number of bins N, which is taken to be the same among layers. The obtained Top-1 accuracy are plotted as a function of N in Fig.5.2.

The accuracy improves as N increases, and becomes comparable to the FP32 result at $N = 16$ (4 bits). Comparing with the result shown in Fig. 4.6, where the same VBQ is applied on all layers, the layer-wise optimization is shown to be important to reproduce the FP32 accuracy; this can only be achieved by combining Eq. 4.2 and SA.

In the experiments so far, I used the same set of 5000 images for the SA optimization and the final evaluation. It is more practical, however, to use a different set of images for the final evaluation. Minimum number of images required for the optimization is also concern, because it is directly proportional to the optimization cost.

In Fig. 5.3, the achieved Top-1 accuracy are plotted as a function of the number of images used for the SA optimization. The Top-1 accuracy are evaluated by using a different set of 5000 images, and the number of bins is taken to be $N = 16$. The figure indicates that the optimization can be performed with as small as 200 images, and no further improvement is expected after 1000 images.

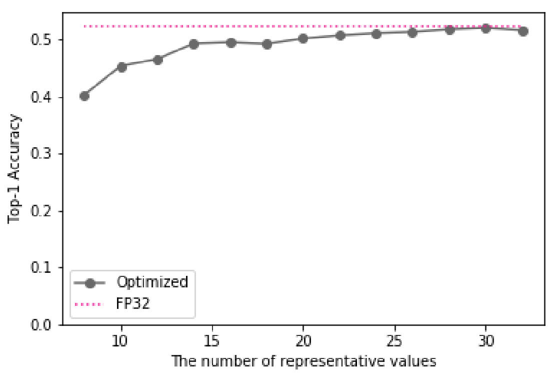


Figure.5.4: Top-1 accuracy of AlexNet using various number of representative values

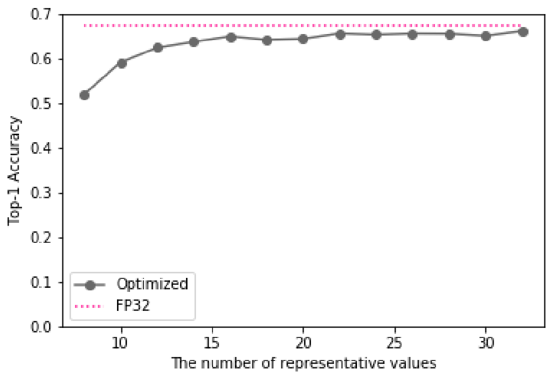


Figure.5.5: Top-1 accuracy of VGG16 using various number of representative values

Table.5.1: Inference Accuracy(Top-1/Top-5) of AlexNet and VGG16 on ImageNet using various model quantization methods

	Model	Decrease of Accuracy (Top-1 / Top-5)	Bit widths Weights	Bit widths Activation	Acceleration
Miyashita et al [13]	AlexNet	- / 6.2%	Conv 5bit(Log2 Format) + FC 4bit	4bit	-
		- / 1.7%	Conv 5bit(Log $\sqrt{2}$ Format) + FC 4bit	4bit	-
	VGG16	- / 6.1%	Conv 5bit(Log2 Format) + FC 4bit	4bit	-
		- / 0.5%	Conv 5bit(Log $\sqrt{2}$ Format) + FC 4bit	4bit	-
Kamiya et al [15]	AlexNet	1.7% / 1.2%	6bit	6bit	1.79
	VGG16	- / 2.16%	6bit	6bit	2.07
Proposed	AlexNet	0.34% / 0.54%	5bit	- (32bit)	1.0
		2.88% / 1.84%	4bit	- (32bit)	1.0
	VGG16	1.36% / 1.26%	5bit	- (32bit)	1.0
		2.54% / 1.50%	4bit	- (32bit)	1.0

5.3 Evaluation

To check the generality of the proposed approach, I have applied our method to AlexNet [34]. AlexNet consists of five convolutional layers and three fully connected layers. It is used for evaluation in many previous studies [13, 15]. I refer [35] for the implementation and the trained data. The Top-1 accuracy at FP32 is 0.5244 in our environment.

The experiments are performed by varying the number of bins N. The SA optimization are performed for VGG16 and AlexNet by using 1000 images, which are evaluated with a different set of 5000 images. The results are shown in Figs. 5.4 and 5.5. AlexNet and VGG 16 show the similar tendency: accuracy becomes significantly worse as N dropped below 16 (4 bits), while it reaches FP32-equivalent.

We compared the inference accuracy of proposed method with other two methods for compressing already-trained models [13, 15]. Different from our work, these prior works reduced bit widths of activation as well as those of weights.

Miyashita et al reduced the weight of the convolutional layer to 5 bits, the weights of the fully connected layer to 4 bits, and the activation to 4 bits. The drop of the top-5 accuracy was 1.7% in AlexNet and 0.5% in VGG16. I succeeded in reducing the weight of all layers to 4 bits with decreasing top-5 accuracy by 1.84% in AlexNet and 1.5% in VGG16 although I still used FP32 for the activation. Kamiya et al reduced the bit width to 6 bits for both weight and activation, and they succeed in accelerating the calculation 1.79 times in AlexNet and 2.07 times in VGG16. In our work, the required amount of computation has not been changed because calculations are performed in FP32. Calculation with the reduced bit width in a dedicated hardware is our future work.

Chapter 6

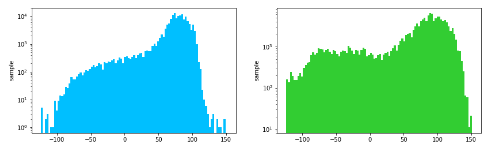
Quantizing Activation

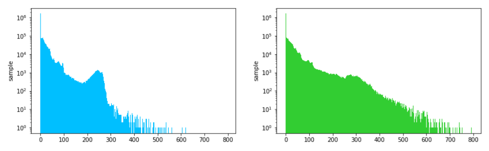
6.1 Histograms of Activation

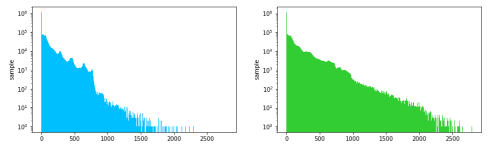
In previous chapters, I have studied a method to quantize weights. In this chapter, I study the method of quantizing activation.First, I investigated the distributions of activation. I used Figure.6.1 from ILSVRC2012 validation dataset and showed the histograms in Figure.6.2 and Figure.6.3. Figure.6.2 shows the histograms of input and the activation of convolutional layers and Figure.6.3 shows the histograms of the activation of fully connected layers. “Input” (Fig6.2(a)) is s matrix subtracted $[103.939, 116.779, 123.68]$, which is the average value of RGB of the data set of the ILSVRC2012, from the input image (Figure.6.1) as pre-process. The y axis of all histograms is log scale. Please note that the scale is different for each layer.

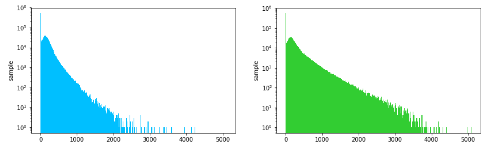
As can be seen from the figure, all activation values are greater than or equal to 0 in all layers since VGG 16 uses ReLU function as an activation function. It is found that the distribution of activation at the middle layer does not different in each image although the distribution of the input varies. I compared the range of distribution of each layers except final layer. The reason why I excepted final layer was because the distribution is always [0, 1] since the softmax function is used as the activation function in the final layer. I found that the distribution ranges significantly varies from 10^1 to 10^4 depending on layer. In the case of weight, even if the same quantization is used for all layers, the answer did not so decrease,

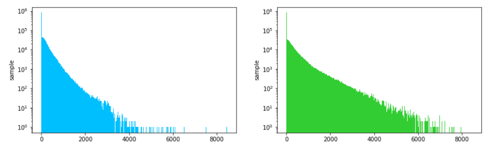
Figure.6.1: Sample Images (ILSVRC2012)

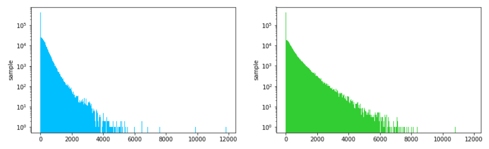

(a) Input

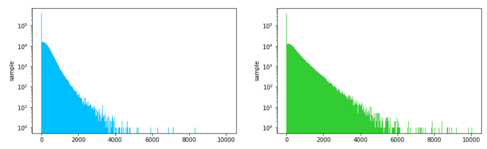

(b) Layer1

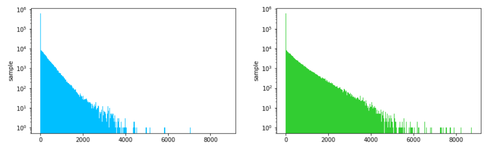

(c) Layer2

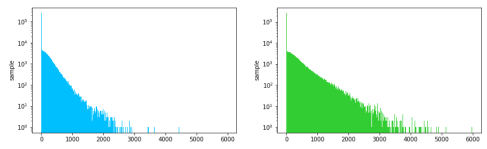

(d) Layer3

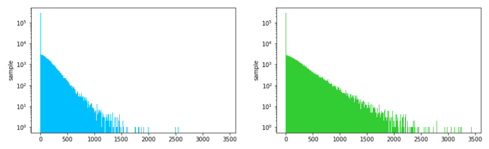

(e) Layer4

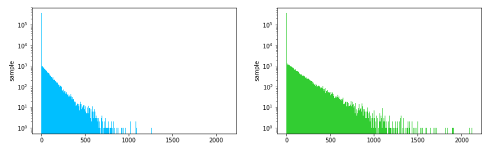

(f) Layer5

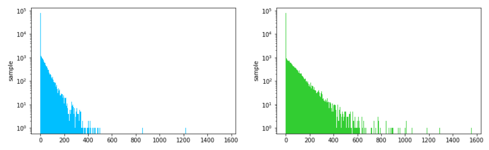

(g) Layer6

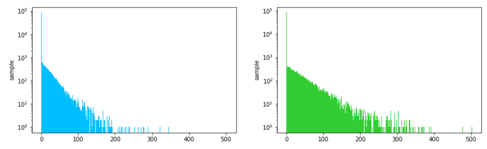

(h) Layer7


(i) Layer8


(j) Layer9


(k) Layer10


(l) Layer11


(m) Layer12

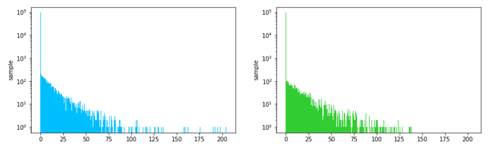
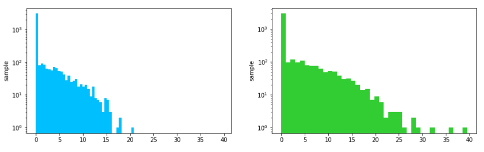
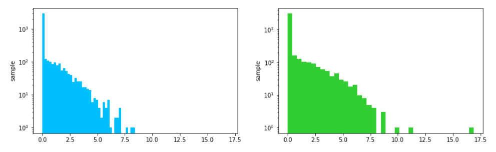

(n) Layer13

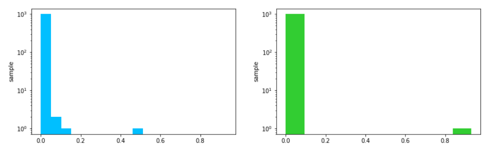
Figure.6.2: Histograms of input images and activation of convolution layers



(a) layer14



(b) layer15



(c) layer16

Figure.6.3: Histograms of activation of fully connected layers

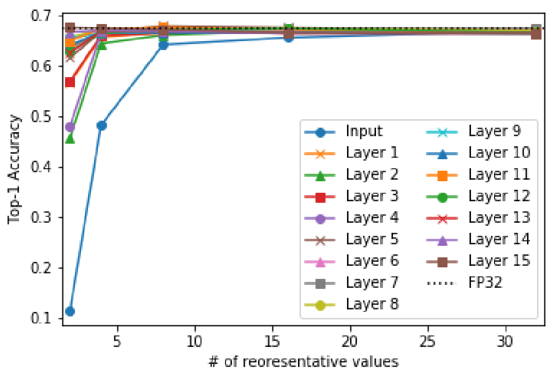


Figure.6.4: Top-1 accuracy of quantizing each layers with various bit width.

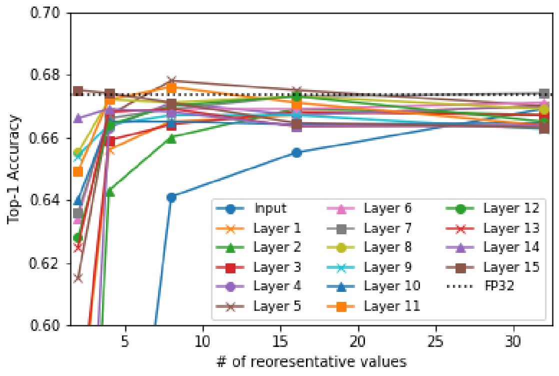


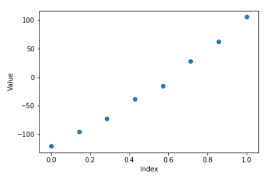
Figure.6.5: The detail of top-1 accuracy of quantizing each layers with various bit width.

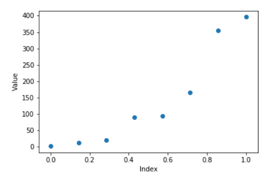
but in the case of activation, using same quantization will lead to markedly decrease since the range of values is too different for each layer. For the initial values when optimizing as well, in the case of weighting, the same values are used in each layer, but in the case of activation, it is necessary to use appropriate values for each layer.

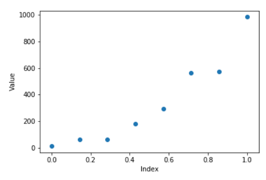
6.2 Optimizing Quantization

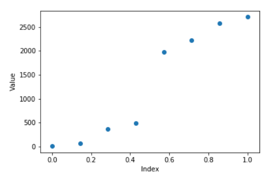
I optimized the position of represent values. First, the relationship between bit width and accuracy is shown in Figure.6.4. Also, the graph which the range of accuracy is only from 0.6 to 0.7 is shown in Figure.6.5. Those figures show that trends differ between input and other layers. It was found that in order to reduce the accuracy drop to 1% or less from FP32, the input layer requires 4 bits, but the other layer requires 1 bit at the minimum and 3 bits at the maximum.

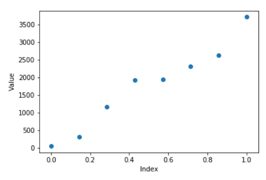
Next, I showed the distribution in case that the number of representative values is 8 in

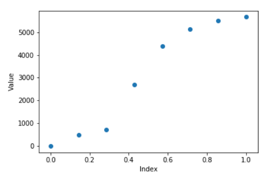

(a) Input

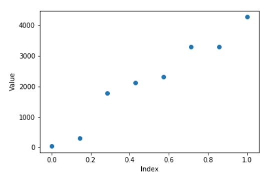

(b) Layer1

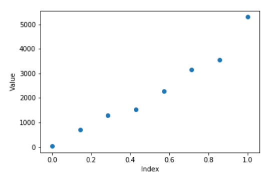

(c) Layer2

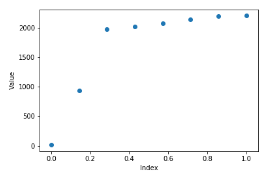

(d) Layer3

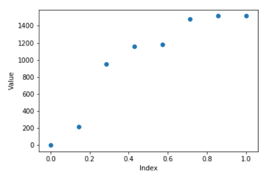

(e) Layer4

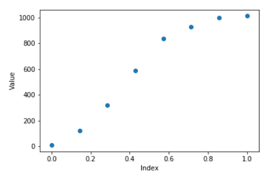

(f) Layer5

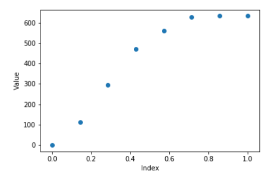

(g) Layer6

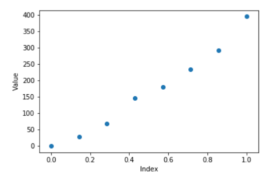

(h) Layer7

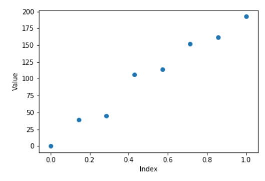

(i) Layer8

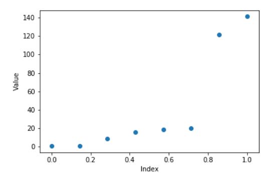

(j) Layer9


(k) Layer10


(l) Layer11


(m) Layer12


(n) Layer13


(o) Layer14

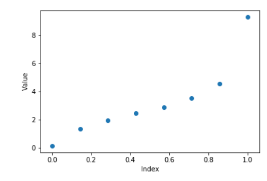

(p) Layer15

Figure.6.6: GA result in 8 represent values(3 bit)

Figure.7.2. I restricted the representative value to a value of 0 or more because the layer other than the input layer takes a value of 0 or more in terms of the structure. On the other hand, I optimized the input layer without any restrictions. I could not find a simple formula regressed distribution of the representative value at the activation.

As shown in Figure.6.6(b)~6.6(p), it can be seen that the 0th representative value is close to 0 in all layers 1 to 15 . Therefore, I guessed that the 0th representative value can be fixed to 0.

A comparison of the accuracy is shown in Figure.6.7 between when the 0th representative value is fixed to 0 and when it is not. There are some layers in which the accuracy is lower than the case where 0 is not fixed, but as a whole, the accuracy improved or almost did not change. Therefore, it was found that there is no problem if the 0th representative value is fixed to 0. By fixing the 0th value to 0, parameters to be optimized are reduced. Based on the results in the figure, the bit width required for the layers excluding input is 1 ~ 3 bits. Therefore, it

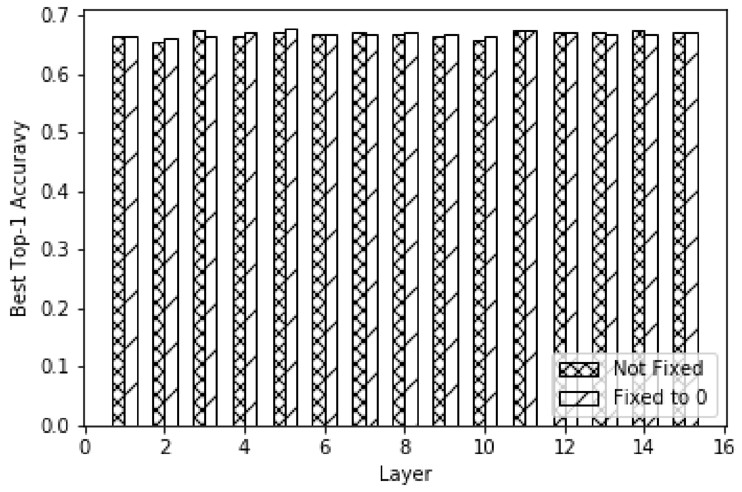


Figure.6.7: The Top-1 Accuracy when 0th representative value is fixed to 0 and when it is not.

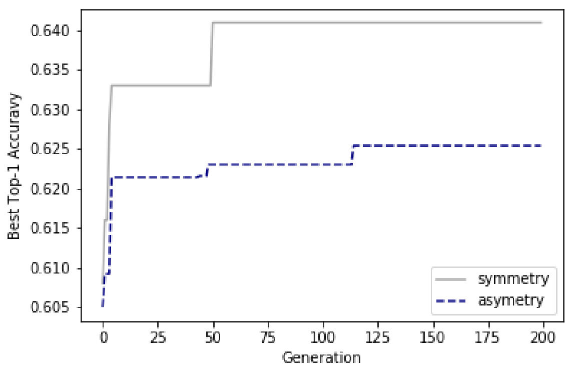


Figure.6.8: The Top-1 Accuracy of Symmetry and Asymmetry Representative Values

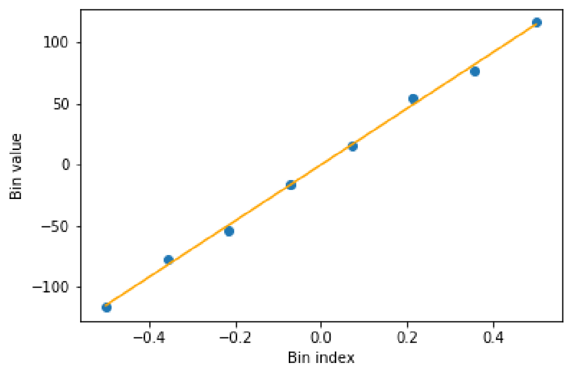


Figure.6.9: Distributions of representative values of input and the regression results by $y = ax$

is enough to optimize $1 \sim 7$ parameters for finding optimum quantization for these layers by fixing the 0th value to 0. This is a sufficiently small number that can simultaneously optimize all the layers.

I focused on the input layer after that. As shown in Figure.6.6(a), the representative values have a linear distribution with almost positive and negative symmetry. Therefore, I optimized representative values for the input layer again by constraining them to be symmetrical relative to origin.

As shown in Figure.6.8, I found that optimization works better when it is constrained so as to be origin symmetrical than when it is not. Moreover, by making origin symmetrical, it is expected that the distribution of representative values of input can be successfully fitted by a straight line passing through the origin. I showed the result of regression by $y = ax$. As the result, I obtained $a = 229.6$ and $R2score = 0.998$. I found that it can be successfully fitted by $y = ax$ even when the representative value is 16 or 32. Therefore, regarding Input, it is optimal to take a representative value based on $y = ax$ and only one parameter a needs to be optimized regardless of the number of representative values.

I reduced the number of parameters to be optimized and now it is possible to do optimization at the same time for all layers. Optimizing those parameters simultaneously is one of my future works.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The size and complexity of CNN models are increasing and as a result they are requiring more computational and memory resources to be used effectively. Use of a lower bit width numerical representation such as binary, ternary or several bit width has been studied extensively so as to reduce the required resources and calculation costs. However, the representation capability of such extremely low bit width is not always sufficient and the accuracy obtained for some CNN models and data is low. There are some prior studies that use moderate lower bit width with well-known numerical representations such as fixed point or logarithmic representation. It is not apparent, however, whether those representations are optimal for maintaining high accuracy.

In this thesis, I studied about the optimal quantization for reducing the bit width in CNN. First, I implemented a framework that allows CNN computation with various numerical expressions in C++ and investigated the distribution of weights in CNN using the framework.

Next, I introduced VBQ for optimizing the quantization method in the weights during CNN inference. I used a genetic algorithm for the optimization method and demonstrated the feasibility of this approach. I discovered that the optimal quantization is different for each network and layer and deduced a heuristic formula for the distribution of the representative values, with which the effort required to optimize VBQ was greatly reduced: I optimized the parameters of the formula using a simulated annealing and showed that higher accuracy can be obtained than in the previous researches.

Lastly, I focused on activation. First, I investigated the distribution of activation and recognized the features. Then, I optimized the quantization using a genetic algorithm and found that in order to reduce the accuracy drop to 1% or less from FP32, the input layer requires 4 bits, but the other layer requires 1 bit at the minimum and 3 bits at the maximum. In terms of the distribution, It is found that as opposed to the distribution of the representative value at the weight, distribution of the representative value at the activation can not be regressed with one simple function for all layers. It was also found that the 0th representative values of layers other than input can be fixed to 0 and the representative values of input layer can be successfully fitted by $y = ax$. By using these two tricks, it is possible to suppressed the number of parameters to the extent that all layers can be optimized simultaneously.

7.2 Future Work

In this thesis, experiments on activation remain at an elementary stage of optimizing quantization for each layer. From the results in this experiment, it was found that the number of parameters can be suppressed to the extent that all layers can be optimized simultaneously. Therefore, optimizing all layers simultaneously is a next step of this studies. And then, based on the result, an experiment that optimize the quantization of both the weight and the activation simultaneously should be conducted.

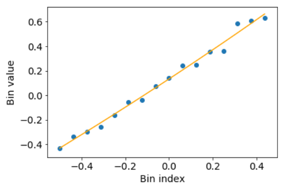
In this experiment, bit width was given as hyper-parameter. However, as I mentioned many times in this paper, it is considered that the required bit width is different for each layer. It is necessary to consider a method that can optimize the quantization and the bit width at the same time.

It is also important to consider an efficient calculation method. Since the optimal numerical expression obtained as a result of my research is different from existing formats such as floating-point and fixed-point, using LUT or designing a dedicated circuit on FPGA will help speed up the calculation.

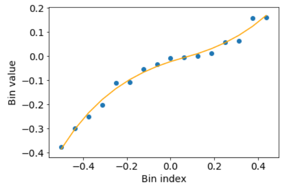
In this paper, I conducted experiments on two kinds of CNN models, VGG16 and AlexNet, but it is not enough to see if the findings obtained in this study apply to CNN in general. Therefore, experiments with other CNN models is useful for obtaining more general knowledge.

Furthermore, the method proposed in this thesis can be applied not only to CNN but also to any machine learning model. Therefore, we can obtain result by applying the proposed method to time-series models such as RNN [36] and LSTM [37] and generative models such as GAN [38] and investigating whether optimal quantization is different depending on kinds of the model.

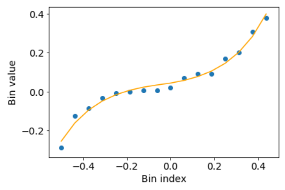
Appendix



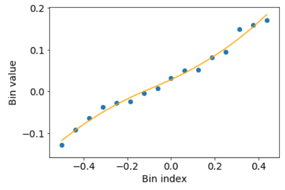
(a) Layer1



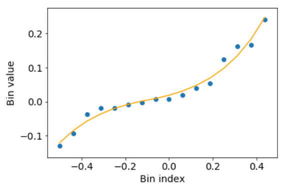
(b) Layer2



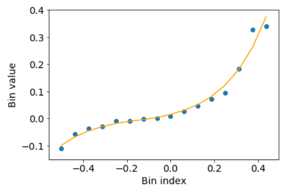
(c) Layer3



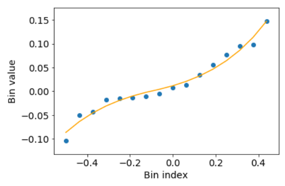
(d) Layer4



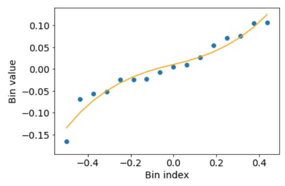
(e) Layer5



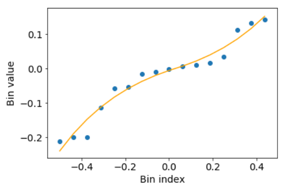
(f) Layer6



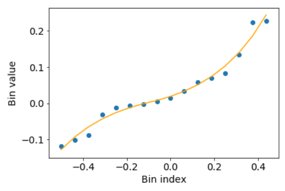
(g) Layer7



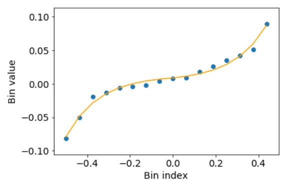
(h) Layer8



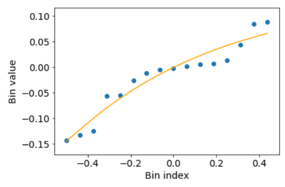
(i) Layer9



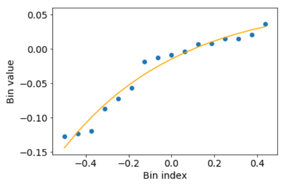
(j) Layer10



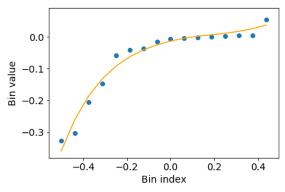
(k) Layer11



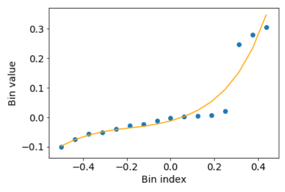
(l) Layer12



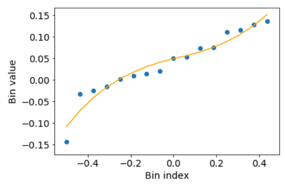
(m) Layer13



(n) Layer14



(o) Layer15



(p) Layer16

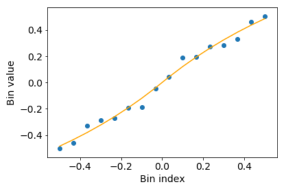
A.7.1: Distributions of representative values and the regression results in the case of asymmetry

Layer	a	b	c	d	R2 score
1	1.010	7.128	-0.350	-5.826	0.990
2	1.222	0.075	-0.007	-1.012	0.988
3	1.424	0.025	0.036	1.207	0.983
4	1.109	0.128	0.009	-0.336	0.989
5	1.311	0.027	0.005	-0.550	0.977
6	1.395	0.017	-0.007	-1.441	0.972
7	1.242	0.028	0.002	-1.432	0.975
8	1.246	0.031	0.010	-0.076	0.952
9	1.193	0.070	0.000	-0.554	0.961
10	1.266	0.037	0.002	-0.583	0.978
11	1.497	0.004	0.008	0.617	0.985
12	1.254	0.027	-0.003	-0.584	0.964
13	1.125	0.044	0.024	3.273	0.969
14	1.355	0.013	0.007	0.077	0.968
15	1.401	0.014	-0.037	-3.964	0.913
16	1.240	0.032	0.052	4.388	0.941

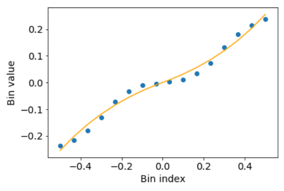
A.7.3: Fitting Parameters in the case of 16 asymmetry representative values in VGG16.

Layer	a	b	R2 score
1	0.313	-1.101	0.987
2	9.001	0.127	0.986
3	37.474	0.049	0.989
4	42.199	0.022	0.993
5	113.925	0.022	0.980
6	39.003	0.068	0.989
7	7.204	0.063	0.994
8	230.705	0.020	0.988
9	15.023	0.041	0.988
10	347.815	0.010	0.968
11	384.525	0.009	0.995
12	41.141	0.020	0.984
13	4.461	0.046	0.992
14	840.920	0.008	0.980
15	43.932	0.061	0.970
16	1.394	0.912	0.972

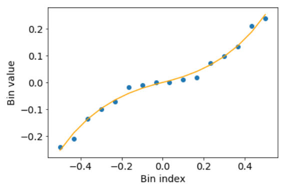
A.7.4: Fitting Parameters in the case of 16 symmetry representative values in VGG16.



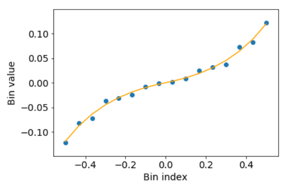
(a) Layer1



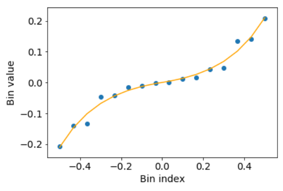
(b) Layer2



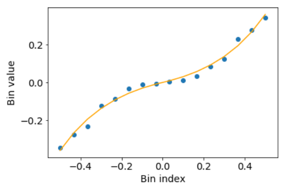
(c) Layer3



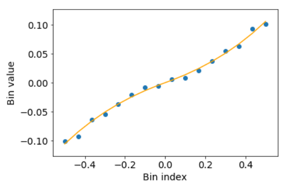
(d) Layer4



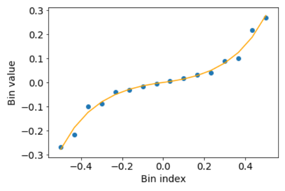
(e) Layer5



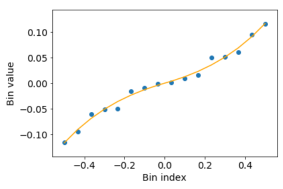
(f) Layer6



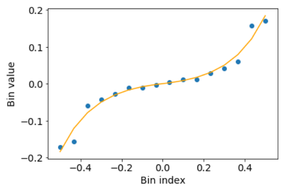
(g) Layer7



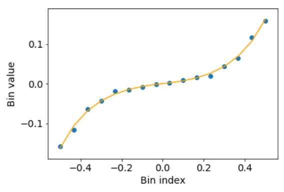
(h) Layer8



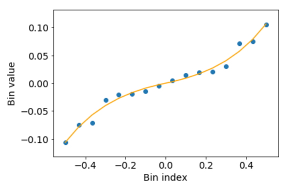
(i) Layer9



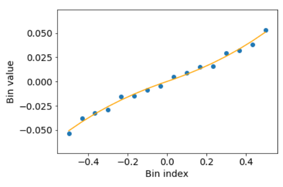
(j) Layer10



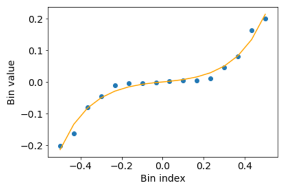
(k) Layer11



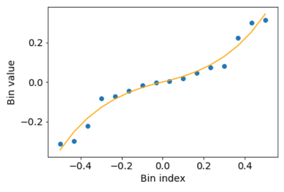
(l) Layer12



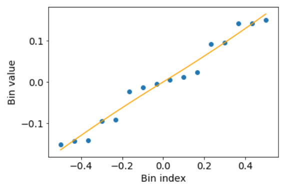
(m) Layer13



(n) Layer14



(o) Layer15



(p) Layer16

A.7.2: Distributions of representative values and the regression results in the case of symmetry

Acknowledgment

My deepest appreciation goes to Prof.Kudoh who provided helpful comments and suggestions. I am also indebted to Dr.Ikegami, Dr.Takano and Dr.O'uchi for their technical help and sincere support. I would also like to express my gratitude to my lab members and my family for their moral support and warm encouragements.

Finary, this paper is based on results obtained from a project by the New Energy and Industrial Technology Development Organization (NEDO).

Reference

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks,” in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA ’15. New York, New York, USA: ACM Press, 2015, pp. 161–170.

[3] J. Qiu, S. Song, Y. Wang, H. Yang, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, and N. Xu, “Going Deeper with Embedded FPGA Platform for Convolutional Neural Network,” in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA ’16. New York, New York, USA: ACM Press, 2016, pp. 26–35.

[4] T. Akiba, S. Suzuki, and K. Fukuda, “Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes,” ArXiv:1711.04325, 11 2017.

[5] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, “ImageNet Training in Minutes,” in Proceedings of the 47th International Conference on Parallel Processing - ICPP 2018. New York, New York, USA: ACM Press, 2018, pp. 1–10.

[6] H. Mikami, H. Suganuma, P. U-chupala, Y. Tanaka, and Y. Kageyama, “ImageNet/ResNet-50 Training in 224 Seconds,” ArXiv:1811.05233, 11 2018.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 6 2016, pp. 770–778.

[8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” Proceedings of the 32nd International Conference on Machine Learning - Volume 37, pp. 1737–1746, 2015.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training Deep Neural Networks with binary weights during propagations,” in Advances in Neural Information Processing Systems 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3123–3131.

[10] Rastegari Mohammad, , V. Ordonez, and Redmon Joseph, and and Farhadi Ali, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks,”

in Computer Vision -ECCV 2016, Leibe Bastian, , J. Matas, and Sebe Nicu, and and Welling Max, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[11] Y. G. L. X. Y. C. Aojun Zhou Anbang Yao, “Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights,” in International Conference on Learning Representations,ICLR2017, 2017.

[12] S.-i. O’uchi, H. Fuketa, T. Ikegami, W. Nogami, T. Matsukawa, T. Kudoh, and R. Takano, “Image-Classifier Deep Convolutional Neural Network Training by 9-bit Dedicated Hardware to Realize Validation Accuracy and Energy Efficiency Superior to the Half Precision Floating Point Format,” in 2018 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 5 2018, pp. 1–5.

[13] D. Miyashita, E. H. Lee, and B. Murmann, “Convolutional Neural Networks using Logarithmic Data Representation,” ArXiv:1603.01025, 3 2016.

[14] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, “Fixed Point Quantization of Deep Convolutional Networks,” in Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ser. ICML’16. JMLR.org, 2016, pp. 2849–2858.

[15] R. Kamiya, T. Yamashita, M. Ambai, I. Sato, Y. Yamauchi, and H. Fujiyoshi, “Binary-decomposed DCNN for accelerating computation and compressing model without retraining,” in Workshop on International Conference on Computer Vision, 2017.

[16] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” 7 2012.

[17] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in International Conference on Learning Representations, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” 9 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[19] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, “Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks,” in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA ’17. New York, New York, USA: ACM Press, 2017, pp. 45–54.

[20] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan, “Training Binary Multilayer Neural Networks for Image Classification using Expectation Backpropagation,” ArXiv:1503.03562, 3 2015.

[21] M. Kim and P. Smaragdis, “Bitwise Neural Networks,” ArXiv:1601.06071, 1 2016.

[22] D. Soudry, I. Hubara, and R. Meir, “Expectation Backpropagation: Parameter-free Training of Multilayer Neural Networks with Continuous or Discrete Weights,” in Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1, ser. NIPS’14. Cambridge, MA, USA: MIT Press, 2014, pp. 963–971.

[23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized Neural

Networks," in Advances in Neural Information Processing Systems 29, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4107–4115.

[24] P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, and D. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," ArXiv:1606.01981, 6 2016.

[25] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive Quantization for Deep Neural Network," Proceedings of AAAI, 2018.

[26] S. Khoram and J. Li, "Adaptive Quantization of Neural Networks," in International Conference on Learning Representations, 2018.

[27] S. Shin, Y. Boo, and W. Sung, "Fixed-point optimization of deep neural networks with adaptive step size retraining," in 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 3 2017, pp. 1203–1207.

[28] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in NIPS Deep Learning and Representation Learning Workshop, 2015.

[29] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in International Conference on Learning Representations, 2018.

[30] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Neural Networks with the Hashing Trick," in Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ser. ICML'15. JMLR.org, 2015, pp. 2285–2294.

[31] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," International Conference on Learning Representations (ICLR), 2016.

[32] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing." Science (New York, N.Y.), vol. 220, no. 4598, pp. 671–80, 5 1983.

[33] V. Černý, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," Journal of Optimization Theory and Applications, vol. 45, no. 1, pp. 41–51, 1 1985.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[35] Michael Guerzhoy and Davi Frossard, "AlexNet implementation + weights in TensorFlow," 2016. [Online]. Available: https://www.cs.toronto.edu/~guerzhoy/tf_alexnet/

[36] J. S. Bridle, "Alpha-nets: A recurrent 'neural' network architecture with a hidden Markov model interpretation," Speech Communication, vol. 9, no. 1, pp. 83–92, 2 1990.

[37] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 11 1997.

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

Publications

[1] S.-i.O'uchi, H.Fuketa, T.Ikegami, W.Nogami, T.Matsukawa, T.Kudoh, and R.Takano. "Image-Classifier Deep Convolutional Neural Network Training by 9-bit Dedicated Hardware ro Realize Validation Accuracy and Energy Efficiency Superior to the Half Precision Floating Point Formar", in 2018 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 5 2018, pp. 1-5.

[2] W.Nogami, T.Ikegami, S.-i.O'uchi, R.Takano, Y.Kishi, and T.Kudoh. "Optimization of Numerical Expression in CNN using Genetic Algorithm", in 2018 Summer United Workshops on Parallel Distributed and Cooperative Processing (SWoPP). 2018-ARC-232 27, 7 2018, pp. 1-6. （優秀若手発表賞）

[3] Y.Kishi, T.Ikegami, S.-i.O'uchi, R.Takano, W.Nogami, and T.Kudoh. "Quantization Optimization for Training of Generative Adversarial Network", in 2018 Summer United Workshops on Parallel Distributed and Cooperative Processing (SWoPP). 2018-ARC-232 13, 7 2018, pp. 1-6.

[4] W.Nogami, T.Ikegami, S.-i.O'uchi, R.Takano, and T.Kudoh. "Optimizing Weight Value Quantization for CNN Inference", in The International Joint Conference on Neural Networks (IJCNN), 2019. (Under Review)

[5] Y.Kishi, T.Ikegami, S.-i.O'uchi, R.Takano, W.Nogami, and T.Kudoh. "Perturbative GAN: GAN with Perturbation Layers", in Thirty-sixth International Conference on Machine Learning (ICML), 2019. (Under Review)