Binary-decomposed DCNN for accelerating computation and compressing model without retraining

Ryuji Kamiya
Chubu University

Takayoshi Yamashita
Chubu University

Mitsuru Ambai
Denso IT Laboratory

Ikuro Sato
Denso IT Laboratory

Yuji Yamauchi
Chubu University

Hironobu Fujiyoshi
Chubu University

Abstract

Recent trends show recognition accuracy increasing even more profoundly. Inference process of Deep Convolutional Neural Networks (DCNN) has a large number of parameters, requires a large amount of computation, and can be very slow. The large number of parameters also require large amounts of memory. This is resulting in increasingly long computation times and large model sizes. To implement mobile and other low performance devices incorporating DCNN, model sizes must be compressed and computation must be accelerated. To that end, this paper proposes Binary-decomposed DCNN, which resolves these issues without the need for retraining. Our method replaces real-valued inner-product computations with binary inner-product computations in existing network models to accelerate computation of inference and decrease model size without the need for retraining. Binary computations can be done at high speed using logical operators such as XOR and AND, together with bit counting. In tests using AlexNet with the ImageNet classification task, speed increased by a factor of 1.79, models were compressed by approximately 80%, and increase in error rate was limited to 1.20%. With VGG-16, speed increased by a factor of 2.07, model sizes decreased by 81%, and error increased by only 2.16%.

1. Introduction

Deep Convolutional Neural Networks (DCNN) realize extremely high recognition accuracy for various tasks such as general object recognition[17], detection[6][13] and semantic segmentation[10][18]. Since AlexNet[11] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012, which involved computation of 1,000 categories, network models with large numbers of layers began to appear, including VGG-16[16] and Residual Network (ResNet)[8], producing remarkable increases in speed and dropping error rates. Recently, there is a continuing trend toward increasing recognition accuracy by increasing the number of layers, as was done with VGG-16 and ResNet. As recognition accuracy increases in this way, models are becoming more complex, and increasing computation time and model size is becoming an issue. Table 1 shows the amount of computation and model sizes for AlexNet, VGG-16 and ResNet, which are de facto standards for DCNN network models. The network model for AlexNet, which won ILSVRC 2012, is composed of five convolutional layers and three fully-connected layers. The network model for VGG-16, which placed second in ILSVRC 2014, is composed of 13 convolutional layers and three fully-connected layers. The network model for ResNet, which won ILSVRC 2015, is composed of 152 convolutional layers and one fully-connected layer. Table 1 shows that convolutional layers accounted for more than 90% of computation in AlexNet, and more than 99% in VGG-16 and ResNet. Although fully-connected layers accounted for roughly 90% of the AlexNet and VGG-16 model sizes, ResNet had roughly the same model size as AlexNet, even though it had 152 layers. This shows that the convolutional layers contribute more to the amount of computation, and the fully connected layers contribute more to model size. It is imperative to increase the

Table 1. Comparison of DCNN and Binarized-DCNN

Model	Computations [million]				Model size [MB]	
	Convolutional		Fully connected		Convolutional	Fully connected
ImageNet classification task						
AlexNet	10.77		0.55		14.29	237.91
VGG-16	153.47		1.26		56.12	471.63
ResNet	113.96		0.021		223.90	7.81
Proposed (AlexNet)	AND	6.06	AND	0.31	2.71	42.14
	bitcount	6.06	bitcount	0.31		
	multiply	0.24	multiply	0.003		
Proposed (VGG-16)	AND	86.33	AND	0.70	10.62	88.64
	bitcount	86.33	bitcount	0.70		
	multiply	4.88	multiply	0.0033		
Proposed (ResNet)	AND	63.67	AND	0.012	43.22	1.49
	bitcount	63.67	bitcount	0.012		
	multiply	7.93	multiply	0.00006		
Places205 scene recognition task						
AlexNet	10.77		0.55		14.29	211.20
VGG-16	153.47		1.20		56.12	459.20
Proposed (AlexNet)	AND	6.06	AND	3.12	2.71	39.79
	bitcount	6.06	bitcount	3.12		
	multiply	0.24	multiply	0.003		
Proposed (VGG-16)	AND	6.06	AND	3.12	10.62	71.91
	bitcount	6.06	bitcount	3.12		
	multiply	0.24	multiply	0.003		

speed of recognition processing and to decrease model size in order to use these methods in environments with limited resources, such as embedded and mobile devices. To resolve this issue, research on accelerating computation and compressing model sizes has been proposed.

BinaryNet[4], Binarized Neural Networks[9] and XNOR-Net[12] are proposed methods to simultaneously accelerate processing and reduce memory use by binarizing DCNN. Both networks express activation values and weights as binary values and express parameters as single bits to reduce memory size and perform high-speed inner products. These methods are able to increase speed and effectively decrease memory use, as is needed, but they require retraining, so they cannot be applied to existing network models. As such, our research proposes Binary-decomposed DCNN, which is able to accelerate inference computation and compress model size for existing network models, without requiring retraining. Binary-decomposed DCNN accelerates recognition processing and compresses models for existing network models by converting feature maps and weightings, which are used in recognition computations in each layer, to binary values and using approximate inner-product computations. The contributions of this method are as follows:

1. Simultaneously accelerates recognition computation and compresses models without the need for retraining by using many binary values and a small number of real values to approximate real-valued parameters.

2. Converts real-valued feature maps to binary feature maps in real time by introducing a quantization sublayer.

3. Can be applied to large-scale network models without great loss of accuracy, unlike BinaryNet and XNOR-Net.

2. Related work

VGG-16 and ResNet achieved high recognition performance in ILSVRC, but models with many layers are complex, so long computation times and large model sizes were issues. To use such models in environments with limited resources, such as embedded devices and smartphones, it will be essential to accelerate recognition processing and compress model sizes. Various research has proposed ways to accelerate processing and compress models, solving these issues.

2.1. Compressing model size by eliminating parameters

Deep Compression[7] and SqueezeNet[2] are research on compressing model size. Deep Compression combines branch pruning, quantization and Huffman encoding to compress model size by approximately 1/50, while increasing performance. It first eliminates connections in a trained model by setting all weight below a certain threshold to zero. This enables the dense weight matrix to be handled as a sparse matrix. The model can then be effectively compressed further using storage methods such as Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC). Similar weights can also be shared by applying clustering to the sparse weight matrix. Finally, the model is compressed using Huffman encoding. The distribution of shared weight indices is uneven so this also improves efficiency of memory use. SqueezeNet introduces Fire modules, compressing models by approximately 1/50 while achieving performance comparable to AlexNet. A Fire module is composed of a Squeeze layer, which replaces 3×3 weight filters with 1×1 weight filters, and an Expand layer, which uses multiple 1×1 and 3×3 weight filters. Using the Squeeze layer at the first stage reduces the dimensionality of the weight filters, and reduces the number of channels needed in the Expand layer. SqueezeNet uses schemes such as introducing the Expand layer and down-sampling in the lower layers preserve inference performance.

2.2. Acceleration and model compression using binary parameters

BinaryNet[4], Binarized Neural Networks[9] and XNOR-Net[12] are methods that simultaneously accelerate computation and compress memory use by Binarizing DCNN feature maps and weights. BinaryNet expresses activation and weight values as binary values, expressing parameters as single bits to reduce memory size, and enabling fast inner product computations. BinaryConnect[5] is used to binarize activation values and weights. When updating parameters, real-valued parameters rather than the binarized parameters are updated. Although binarizing activation and weight values achieves both faster computation and model compression, parameters cannot be updated through back-propagation of error. As such, BinaryNet computes updated weights by replacing some parameters only when performing parameter clipping and gradient computation. XNOR-Net improves on the accuracy of BinaryNet by introducing scaling coefficients. To do so, both binary filters and scale factors are approximated to minimize the approximation error due to each weights in the BinaryConnect binarization method. To compute the parameter updates, a method similar to BinaryNet is used. There are also regions during convolution computations where inner product computations are duplicated. This efficiency is improved by computing the average absolute value in the channel direction and convolving that output with the binary filter.

3. Proposed method

Our proposed method simultaneously accelerates inference computation and compresses models for DCNN by transforming feature maps and weights to binary values. The proposed method consists of two parts: (1) decomposing real-valued vector of weights to binary basis vectors, (2) quantization sub-layer.

3.1. Decomposing real-valued vector to binary basis vectors

To use binary inner product operations, real-valued parameters must be converted to binary values. One method for converting parameters to binary is vector decomposition[15][19][1]. Vector decomposition breaks down a weight vector, $\mathbf{w} \in \mathbb{R}^D$, into a binary basis matrix, $\mathbf{M} \in \{-1, 1\}^{D \times k}$, and a scaling coefficient vector, $\mathbf{c} \in \mathbb{R}^k$. Here, k is the number of basis vectors, or basis rank, and D is the input dimensionality. By applying vector decomposition to the weight vectors, inner products between two real values can be replaced with inner products between binary values when an input vector, $\mathbf{x}$, is binarized. Inner products between binary values can be done quickly using logical operations and bit counting. Two algorithms for optimizing vector decomposition are the greedy algorithm[15] and the exhaustive algorithm[19]. In this section, we describe decomposition using an exhaustive optimization algorithm which is better than the greedy algorithm.

3.1.1 Exhaustive algorithm [19]

Decomposition by the exhaustive algorithm computes a binary basis matrix, $\mathbf{M}$, and scaling vector, $\mathbf{c}$, that minimize the cost function in Eqn. 1 on the weight vector, $\mathbf{w}$. The decomposition is very slow, optimizing $\mathbf{M}$ through exhaustive search, but it can provide a decomposition with better approximation performance than the greedy algorithm. The decomposition algorithm is shown in Algorithm 1. First, $\mathbf{M}$ is initialized to random values from $\{-1, 1\}$, and $\mathbf{c}$ is initialized to random real values. Then, $\mathbf{M}$ and $\mathbf{c}$ are optimized. It is difficult to optimize both $\mathbf{M}$ and $\mathbf{c}$ simultaneously, so they are each optimized alternately. $\mathbf{M}$ is fixed, and $\mathbf{c}$ is optimized by minimizing Eqn. 1 using the least squares method. Then, $\mathbf{c}$ is fixed and $\mathbf{M}$ is optimized by exhaustive search. This process is repeated until the value of the cost function, Eqn. 1, converges. Note that the accuracy of approximation of the vector decomposition depends on the initial values, so we take the values of $\mathbf{M}$ and $\mathbf{c}$ that minimize Eqn. 1 after changing the initial values L times as the basis decomposition result.

$$E = ||\mathbf{w} - \mathbf{M}\mathbf{c}||_2^2 \tag{1}$$

Algorithm 1 Decomposition algorithm

Require: $\mathbf{w}$, k, L
  for i to L **do**
    Initialize $\mathbf{M}_i$ by random values on $\{-1, 1\}$
    repeat
      $\mathbf{c}_i = (\mathbf{M}_i^{\mathrm{T}}\mathbf{M}_i)^{-1}(\mathbf{M}_i^{\mathrm{T}}\mathbf{w})$
      $\mathbf{M}_i = \underset{\mathbf{M}_i \in \{-1,1\}^{D \times k}}{\arg\min} ||\mathbf{w} - \mathbf{M}_i\mathbf{c}_i||_2^2$
    until $||\mathbf{w} - \mathbf{M}\mathbf{c}||_2^2$ converges
    $\hat{\mathbf{M}}, \hat{\mathbf{c}} = \underset{\mathbf{M},\mathbf{c}}{\arg\min} ||\mathbf{w} - \mathbf{M}\mathbf{c}||_2^2$
  end for
  return $\hat{\mathbf{M}}, \hat{\mathbf{c}}$

3.1.2 Applying vector decomposition to the convolution layers

First, consider application of vector decomposition to the mth weight filter associated with the nth feature map in the lth layer, $\mathbf{w}_{n,m}^l \in \mathbb{R}^{H \times H}$. Vector decomposition applies to vectors, so it cannot be applied directly to the weight filter, which is a matrix. As such, we apply vector decomposition by expressing the weight filter as a vector. Expressing the weight filter as a vector results in a vector of dimension $H \cdot H$. With network models as in VGG-16 and ResNet, the weight filter for each layer is usually very small, so this does not reduce the amount of computation using approximate inner-product computations has little effect. Thus, rather than decomposing a single weight filter, filters in the channel direction are expressed as a vector. With $\mathbf{M}$ as the number of input feature maps, the decomposed weight vectors can be defined as . Then, the dimension of $\mathbf{W}_n$ is $M \cdot H^2$, so approximate inner-product computations have more effect. In convolutional layers, $\mathbf{W}$ only has N output maps. Vector decomposition using Algorithm 1 is applied to each $\mathbf{W}$, decomposing them into and . In convolutional layer inference processing, these and are used for approximate inner product calculations.

3.1.3 Applying vector decomposition to the fully-connected layers

Next, consider application of vector decomposition to connection weights in the nth unit of the lth fully-connected layer, $\mathbf{w}_n^l \in \mathbb{R}^M$. The weights used to get the output from the nth unit of a fully connected layer are an M-dimensional weight vector. Also a fully connected layer has values equaling the number of output units, N. Vector decomposition using Algorithm 1 is applied to each weight vector, $\mathbf{w}_n^l$, to decompose into the binary basis matrix, $\hat{\mathbf{M}}$, and the scaling coefficient vectors, $\hat{\mathbf{c}}_n$.

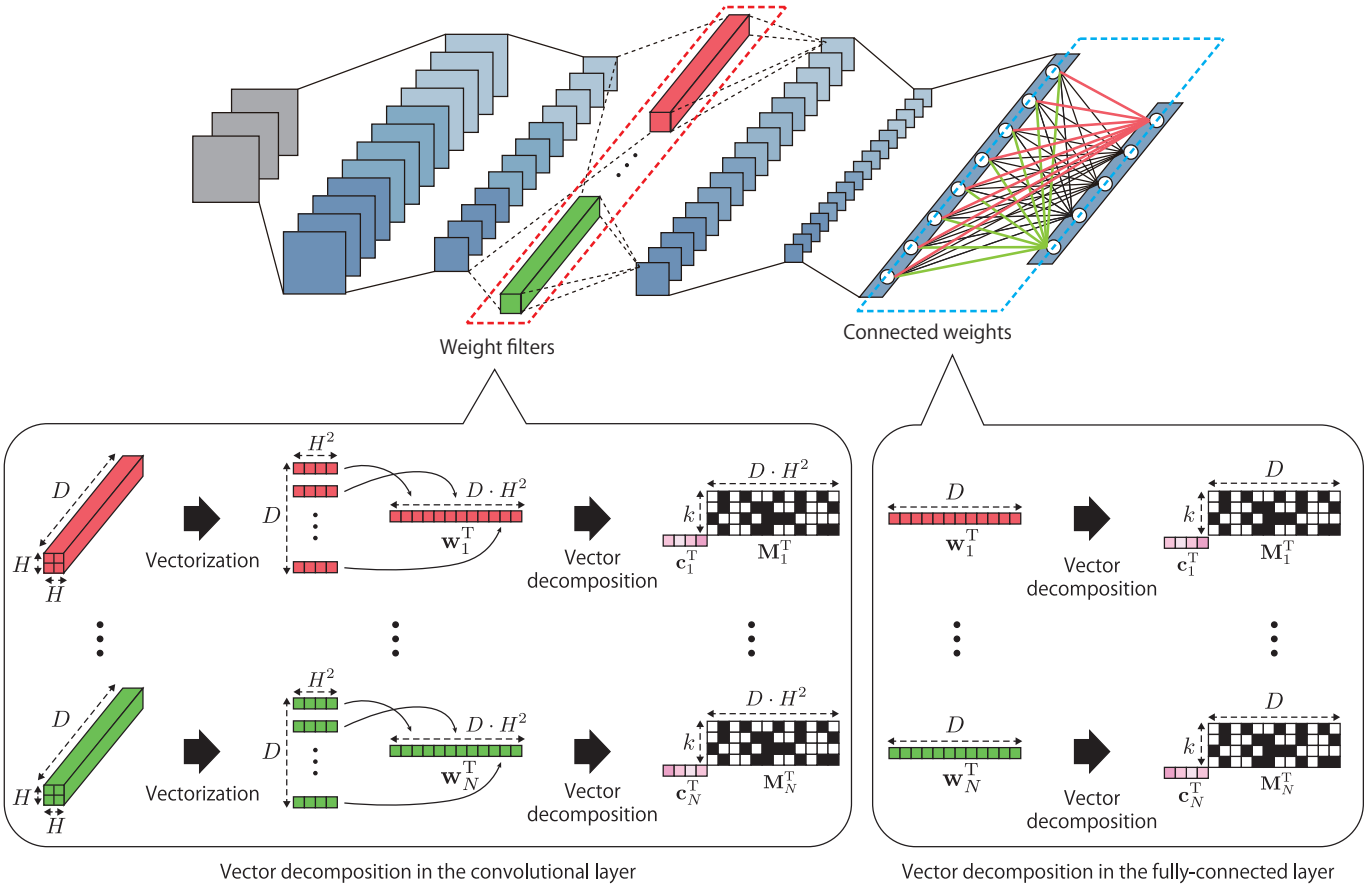


Figure 1. Weights decomposition in DCNN layers

3.2. Quantization sub-layer

Quantization can convert to binary rapidly, but it applies to a fixed range, so real and negative values cannot be quantized. Thus, we introduce a quantization sub-layer able to binarize real values, including negative values, rapidly. The quantization sub-layer is able to quantize real values including negative values by changing the quantization range.

Before quantizing a feature map, the quantization bit-depth, Q, is decided. The approximation accuracy of quantization increases with larger Q, but the amount of computation required for approximate inner products also increases, making computation slower. Conversely, as Q becomes smaller, the amount of computation decreases, so inference computations become faster, but approximation accuracy decreases due to quantization. First, Eqn. (2) is used to find the quantization range, Δd, between the maximum and minimum values in feature map. Δd depends on the maximum and minimum values in the feature map, so the value is different for each feature map.

$$\Delta d = \frac{\max(\mathbf{x}) - \min(\mathbf{x})}{2^Q - 1} \tag{2}$$

Next, Eqn. (3) is used to shift the minimum value of the feature map to 0. Here, $\mathbf{1}$ represents the unit vector. Shifting the feature map enables quantization of even negative values, which normally could not be quantized.

$$\mathbf{x}' = \frac{\mathbf{x} - \mathbf{1}\min(\mathbf{x})}{\mathbf{1}Q} \tag{3}$$

Finally, $\mathbf{x}'$ is quantized. Quantizing $\mathbf{x}'$ generates a binary code, $\mathbf{B} \in \{0,1\}^{D \times Q}$. The binary code from the quantization sub-layer can be recovered using Eqn. 4.

$$\mathbf{x} \approx \mathbf{B}\mathbf{r} + \mathbf{1}\min(\mathbf{x}) \tag{4}$$

3.3. Inference processing

Binary-decomposed DCNN accelerates forward computation of network using approximate binary inner product computations. To perform operations on two binary values, the quantization sub-layer is introduced to binarize feature maps $\mathbf{B}$. Feature map values and feature vectors in each layer depend on the input samples, so in contrast to vector decomposition of the weights, quantization cannot be done ahead of time. The ability of the quantization sub-layer to rapidly binarize real values is used to perform this quantization during inference computations.

3.3.1 Computation in convolution layers

For ordinary convolution computations, the n-th feature map $u_{n,i,j}$ is obtained as the sum of products of the local area of feature mapping $\mathbf{x}_{i,j}$, and weight filter $\mathbf{w}_n^{\mathrm{T}}$. In our method the output is computed by replacing the sum of products of real values with binary operations. Input feature maps are first quantized by the quantization sub-layer. This generates binary feature maps, $\mathbf{B} \in \{0,1\}^{MH^2 \times Q}$. Then the convolution is computed using the binary feature maps and binary weight filters, $\mathbf{M}_n^{\mathrm{T}}$ and $\mathbf{c}_n^{\mathrm{T}}$, obtained through vector decomposition. The output $u_{i,j,n}$ is given by Eqn. 5.

$$\begin{aligned} u_{n,i,j} &= \mathbf{w}_n^{\mathrm{T}} \mathbf{x}_{i,j} \\ &\approx \hat{\mathbf{c}}_n^{\mathrm{T}} \hat{\mathbf{M}}_n^{\mathrm{T}} (\mathbf{B}_{i,j} \mathbf{r}_{i,j} + \mathbf{1} \min(\mathbf{x})) \\ &= \hat{\mathbf{c}}_n^{\mathrm{T}} \hat{\mathbf{M}}_n^{\mathrm{T}} \mathbf{B}_{i,j} \mathbf{r}_{i,j} + \hat{\mathbf{c}}_n^{\mathrm{T}} \hat{\mathbf{M}}_n^{\mathrm{T}} \mathbf{1} \min(\mathbf{x}) \end{aligned} \quad (5)$$

Here, $\mathbf{w}_n^{\mathrm{T}} \in \mathbb{R}^{DH^2}$ is the weighting used when generating the n-th feature map, and $\mathbf{x}_{i,j} \in \mathbb{R}^{DH^2}$ is the feature map used when generating the unit at coordinates i, j in the output feature map.

3.3.2 Computation in fully-connected layers

The i-th output u_i, from fully-connected layers are computed by inner products between real-valued feature vectors, $\mathbf{x}$, and connection weights, $\mathbf{w}_i$. To accelerate inner-product calculations in fully-connected layers, real values are replaced with binary values, as in the convolutional layers. This generates binary feature matrix $\mathbf{B} \in \{0,1\}^{D \times Q}$ from the input feature vector $\mathbf{x} \in \mathbb{R}^D$. Then, the output i is approximated by Eqn. 6 from the binary feature matrix and the binary weight vectors decomposed beforehand.

$$\begin{aligned} u_i &= \mathbf{w}_i^{\mathrm{T}} \mathbf{x} \\ &\approx \hat{\mathbf{c}}_i^{\mathrm{T}} \hat{\mathbf{M}}_i^{\mathrm{T}} (\mathbf{B}\mathbf{r} + \mathbf{1} \min(\mathbf{x})) \\ &= \hat{\mathbf{c}}_i^{\mathrm{T}} \hat{\mathbf{M}}_i^{\mathrm{T}} \mathbf{B}\mathbf{r} + \hat{\mathbf{c}}_i^{\mathrm{T}} \hat{\mathbf{M}}_i^{\mathrm{T}} \mathbf{1} \min(\mathbf{x}) \end{aligned} \quad (6)$$

where, $\hat{\mathbf{M}}^{\mathrm{T}} \in \{-1,1\}^{k \times D}$ and $\mathbf{B} \in \{0,1\}^{D \times Q}$ are binary, so it can be computed using logical operators and bit counting, as in Eqn. 7. The computation can be done at high speed, counting bits using the POPCNT function implemented in the Streaming SIMD Extension (SSE) 4.2.

$$\hat{\mathbf{m}}_i^{\mathrm{T}} \mathbf{b}_j = 2 \times \mathrm{POPCNT}(\mathrm{AND}(\hat{\mathbf{m}}_i^{\mathrm{T}}, \mathbf{b}_j)) - \|\mathbf{b}_j\|_1^2 \quad (7)$$

4. Experiments

In testing, we evaluated recognition performance, processing time and model size when applying the proposed method to several network models. Quantization bit depth, Q, of 4, 6 and 8 were used for approximations, and similarly, basis rank of 4, 6, and 8 were used. Quantization bit depths and basis rank less than 4 were not used because error rates increased to a great degree. Similarly, with quantization bit rate and basis rank greater than 8, the drop in error rate had peaked, so they were not included. To evaluate model size in testing, the total memory occupied by the network model, including weights, $\mathbf{W}$, binarized basis matrix, $\mathbf{M}$, and scaling coefficient vector, $\mathbf{c}$, were compared. Published trained models were used as parameters for each network model, with no fine tuning. Top-5 accuracy was used to evaluate recognition performance. Top-5 accuracy is a method that counts cases where the training signal is included among the five most probable inferred classes, as success. We used an Intel Core i7-4770 3.40-GHz processor.

4.1. ImageNet classification task

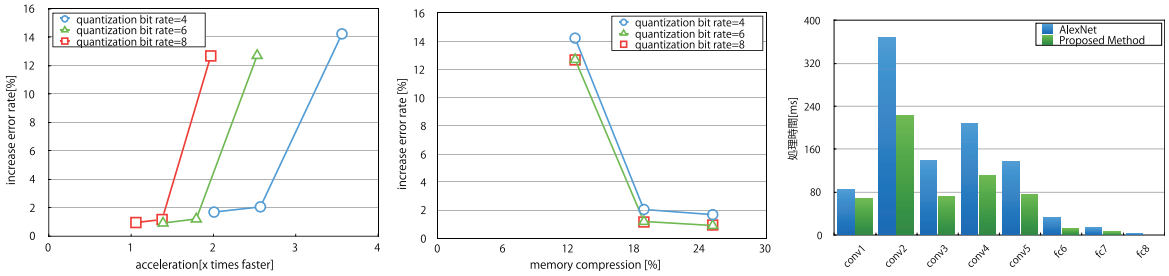
Testing was done using the AlexNet, VGG-16, and ResNet-152 network models. The ImageNet data set used in the ILSVRC 2012 [14] classification task was used. ImageNet is a very large object recognition dataset, containing 1,200,000 training samples, 100,000 test samples, and 50,000 validation samples. Each sample is classified into one of 1000 categories. In testing, evaluation was done using the 50,000 validation samples.

4.1.1 Model 1: AlexNet

AlexNet is composed of 5 convolutional layers and 3 fully-connected layers. A comparison of recognition accuracy, processing time, and error-rate increases when using approximate inner-product calculations is shown in Figure 2. Here, k indicates the basis rank used for weight decomposition. Comparing the same basis rank for quantization bit-depths of 6 and 8, almost no increase in the error rate is shown. In this case, the lower quantization bit-depth of 6 is better, requiring less computation. Figure 2(b) shows that memory compression does not change between quantization bit depths of 4 and 6 using the same basis rank, so we can say quantization bit-depth does not affect reduction of memory use. The error rate also does not change greatly when comparing basis ranks of 6 and 8. This is similar to the trend in Figure 2(a). For both quantization bit-depth and basis rank of 6, speed increased by a factor of 1.79, and model size decreased from 237.91 MB to 44.85 MB. Here, error rates increased by 1.20%.

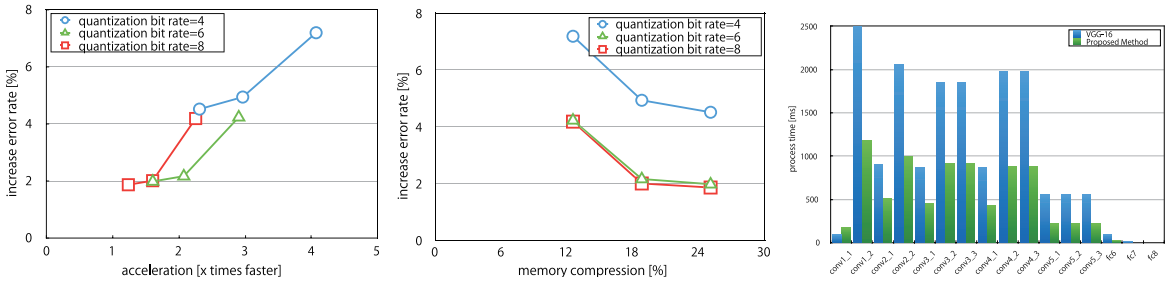
4.1.2 Model 2: VGG-16

VGG-16 is composed of 13 convolutional layers and 3 fully connected layers. The dimensions of the weight filter and number of input feature maps for the first convolutional layer are very small, so approximating inner-product computations may not produce significant effect. For this reason, approximations were introduced on all but the first



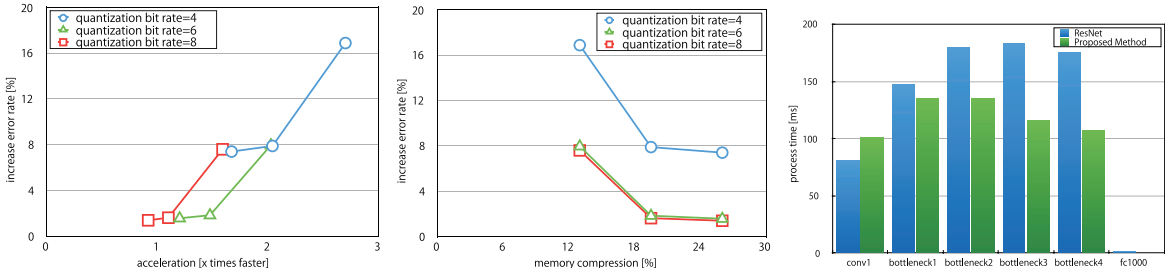
(a) error vs acceleration (b) error vs memory compression (c) process time

Figure 2. ImageNet performance evaluation for AlexNet



(a) error vs acceleration (b) error vs memory compression (c) process time

Figure 3. ImageNet performance evaluation for VGG-16



(a) error vs acceleration (b) error vs memory compression (c) process time

Figure 4. ImageNet performance evaluation for ResNet-152

layer for testing. Recognition accuracy, processing time, and model size with approximate inner-product computations are shown in Figure 3. With a quantization bit-depth of 6 and basis rank of 6, speed increased by a factor of 2.07 and the model size reduced from 527.74 MB to 99.26 MB. In this case, the error rate increased by 2.16%.

4.1.3 Model 3: ResNet-152

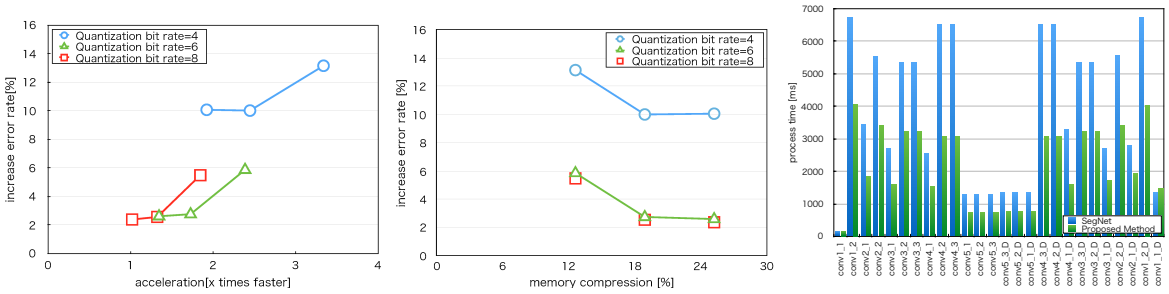
ResNet-152 is composed of 151 convolutional layers and 1 fully connected layer. Recognition accuracy, processing speed and model size using the approximate inner product calculations are shown in Figure 4. ResNet has 152 convolutional layers, which account for approximately 96% of the model size. Clearly, reduction in model size can also be gained for models like ResNet, which have very many convolutional layers. With a quantization bit-depth of 6 and basis rank of 6, speed increased by a factor of 1.77 and the model size reduced from 229.08 MB to 44.71 MB. In this case, the error rate increased by 1.86%.

Table 2. Comparison of performance with methods based on AlexNet

Model	Top1	Top5	acceleration	compression
AlexNet	56.8	80.0	-	-
Deep Compression	56.8	79.9	1.00	35
XNOR-Net	44.2	69.2	58.0	32
proposed	55.1	78.8	1.79	5

4.1.4 Comparison with other methods

In this section, we compare the top-1/5 accuracy, acceleration scale factors and model compression ratios of the proposed method with those of three other methods (AlexNet, Deep Compression and XNOR-Net). The proposed method involves applying vector decomposition to a model with



(a) error vs acceleration (b) error vs memory compression (c) process time

Figure 5. Cityscapes dataset performance evaluation for SegNet

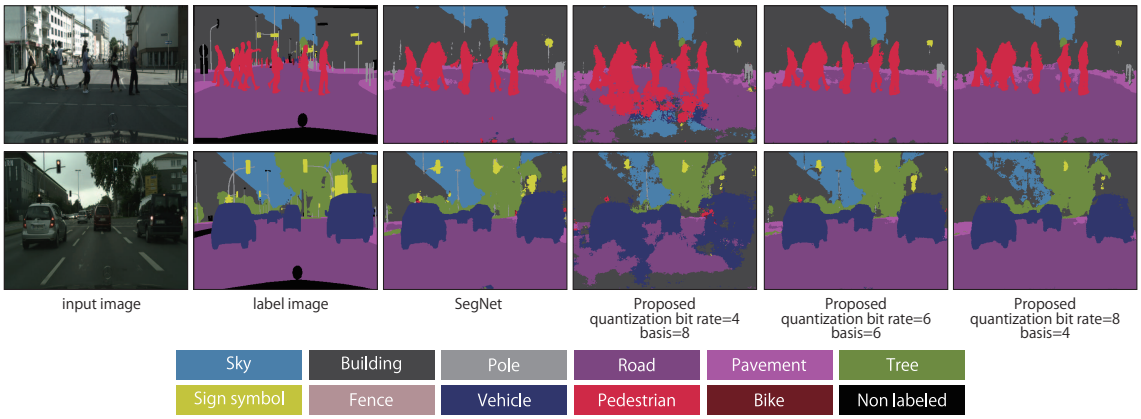


Figure 6. Semantic segmentation results in Cityscapes

a quantization bit depth of 6. The comparison results are shown in Table 2. Although Deep Compression had excellent compression performance, its recognition computation was no faster. XNOR-Net had superior acceleration scaling performance and compression performance, but its recognition accuracy was much lower. The proposed method had worse acceleration scaling and compression performance than the other methods, but maintained its recognition accuracy while simultaneously achieving a high acceleration scale factor and model compression.

4.2. Cityscapes semantic segmentation task

We performed an evaluation of semantic segmentation using the Cityscapes Dataset[3]. The Cityscapes Dataset is a very large segmentation dataset. In this experiment, we performed the evaluation using the 500 verification samples. For the network model, we used SegNet[18]. For the model parameters, we used published learned parameters, and fine tuning was not performed. Figures 5 and 6 show the recognition performance when using Cityscapes. With a quantization bit depth of 6 and a basis rank of 6, we achieved an acceleration scale factor of approximately 1.73 and compressed the model size from approximately 112.25 MB to approximately 21.23 MB. In this case, the rate of error increase was approximately 2.75%.

5. Conclusion

We have proposed a method able to accelerate inference computation while compressing model sizes, using existing network models without the need for retraining. The method compresses memory use by converting weightings from each layer from real-valued parameters to binary parameters, and accelerates inference computation by replacing real-valued inner product calculations with binary valued inner product calculations using logical operations and bit counting. Using a quantization bit-depth of 6 and basis rank of 6, AlexNet model sizes were reduced by approximately 80%, and speed increased by a factor of 1.79. In this case, error rates increased by 1.20%. With VGG-16, model sizes reduced by 81%, and speed increased by a factor of 2.07. In this case, error rates increased by 2.16%. In the future, we plan to increase approximation accuracy and reduce the increases in error rates.

References

[1] M. Ambai and I. Sato. SPADE: Scalar Product Accelerator by Integer Decomposition for Object Detection. In *ECCV*, 2014.

[2] F. N. Candela, M. W. Moskewicz, K. Ashram, S. Han, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy

with 50x fewer parameters and <1MB model size. *arXiv preprint arXiv:1602.02830*, 2016.

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] M. Courbariaux and Y. Bengio. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[5] M. Courbariaux, Y. Bengio, and J. David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015.

[6] R. Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

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

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] I. Hubara, D. Soudry, and R. E. Yaniv. Binarized Neural Networks. *arXiv preprint aiXiv:1602.02505*, 2016.

[10] L. Jonathan, S. Evan, and D. Trevor. Fully Convolutional Networks for Semantic Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105. Curran Associates, Inc., 2012.

[12] R. Mohammad, O. Vicente, R. Joseph, and F. Ali. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *European Conference on Computer Vision (ECCV)*, 2016.

[13] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Neural Information Processing Systems (NIPS)*, 2015.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[15] P. H. T. Sam Hare, Amir Saffari. Efficient Online Structured Output Learning for Keypoint-Based Object Tracking. In *the Proceedings IEEE Conference of Computer Vision and Pattern Recognition (CVPR)*, 2012.

[16] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[18] B. Vijay, K. Alex, and C. Roberto. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[19] Y. Yamauchi, A. Mitsuru, S. Ikuro, Y. Yoshida, and H. Fujiyoshi. Distance Computation Between Binary Code and Real Vector for Efficient Keypoint Matching. *Information Processing Society of Japan Transactions on Computer Vision and Applications (CVA)*, 5:124–128, 2013.