Master's Thesis

Exploration of Divided DCNN for Object Recognition using both Cloud and Edge Computing

Ryuta Shingai
Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Prof. Professor Yasuhiko Nakashima
Computing Architecture Lab. (Division of Information Science)

Submitted on March 17, 2020

A Master's Thesis
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
MASTER of ENGINEERING

Ryuta Shingai

Thesis Committee:

Professor Yasuhiko Nakashima
(Supervisor, Division of Information Science)
Professor Michiko Inoue
(Co-supervisor, Division of Information Science)
Associate Professor Takashi Nakada
(Co-supervisor, Division of Information Science)
Assistant Professor Tran Thi Hong
(Co-supervisor, Division of Information Science)
Assistant Professor Renyuan Zhang
(Co-supervisor, Division of Information Science)

Exploration of Divided DCNN for Object Recognition using both Cloud and Edge Computing*

Ryuta Shingai

Abstract

Modern deep learning has significantly improved the performance and has been used in a wide variety of applications. Since the amount of computation required for the inference process of the neural network is large, it is processed not by the data acquisition location like a surveillance camera but by the server with abundant computing power. As the number of IoT devices increases, traffic congestion and power consumption of server communications are of concern. By processing a part of DCNN on the edge, communication size to be transferred is smaller than the sensor data size. Therefore, I considered dividable network models using edge computing.

In this paper, I have evaluated AlexNet and VGG16 on the divided environment and estimated FPS values with Wi-Fi, 3G, and 5G communication for general object recognition (1000 class). As necessary conditions, I set FPS to 30 or faster and object recognition accuracy to 69.7% or higher. This value is based on that of an approximation network model. I constructed performance and energy models to find the optimal configuration that consumes minimum energy while satisfying the necessary conditions. When dividing in 5th pooling layer using HEVC, the compressed intermediate data can be reduced by 65.5% with AlexNet and 49.7% with VGG16 than the compressed sensor data. From the perspective of FPS, accuracy and energy, I found the following. When using

*Master's Thesis, Graduate School of Science and Technology, Nara Institute of Science and Technology, March 17, 2020.

AlexNet, it is most efficient to execute all inference processing at the edge computing (Energy consumed by Wi-Fi, 3G, and 5G: 0.165 [J/frame]) and the value can be reduced by about 87.9% compared to executing all process in the server. When using VGG16, it is most efficient to execute all the inference processing on the server (Energy consumed by Wi-Fi and 5G: 1.78 [J/frame], Energy consumed by 3G: 1.79 [J/frame]). If the computing power of edge computing is accelerated about 1.36 times, processing can be performed on the edge device up to 2th pooling layer in VGG16, and the energy consumption for inference can be reduced by about 6.74% compared to executing all process in the server.

Keywords:

Deep Neural Network, Edge-computing, Inference, Distributed Network, Video Compression

クラウドとエッジコンピューティングを併用する物体認識のための分割DCNNの調査*

新谷 隆太

内容梗概

最近の深層学習はパフォーマンスを大幅に向上させ、さまざまな場面で利用されている。ディープニューラルネットワークの推論処理に必要な計算量は大きいため、監視カメラなどのようなデータ取得場所ではなく、豊富な計算能力を持つサーバーによって処理される。IoT デバイスの数が増加していく中、サーバーにおける通信のトラフィック輻輳や消費電力が問題として懸念されている。エッジ側でDCNNの一部を処理することにより、転送するデータの通信量はセンサーデータの量よりも小さくなるため、エッジコンピューティングを用いた分割可能なネットワークモデルを検討した。

本稿では、一般物体認識タスク（1000 クラス）に対して、分割環境によるAlexNet と VGG16 を評価し、Wi-Fi、3G および 5G 通信で FPS を推定した。必要な要件として、FPS を 30 以上に設定し、物体認識精度を 69.7%以上に設定した。この値は、近似ネットワークモデルの値に基づいて決定されている。分割可能なモデルを構築して、必要な条件を満たすと同時に最小のエネルギーを消費する最適な構成を発見した。HEVC を使用して 5 番目の pooling 層で分割する場合、圧縮後のセンサーデータよりも圧縮後の中間データのほうが AlexNet だと 65.5%削減でき、VGG16 だと 49.7%削減することがわかった。FPS、精度および消費エネルギーの観点から、次のことが判明した。 AlexNet の場合、すべての推論処理をエッジコンピューティングで実行するのが最も効率的であり (Wi-Fi および 3G、5G での消費エネルギー : 0.165 [J/frame])、その値は、サーバーですべてのプロセスを実行するのに比べて約 87.9 %削減できます。VGG16 を使用する場合、サーバーですべての推論処理を実行するのが最も効率的です (Wi-Fi および 5G での消費エネルギー : 1.78 [J/frame]、3G での消費エネルギー : 1.79 [J/frame])。エッ

*奈良先端科学技術大学院大学 先端科学技術研究科 修士論文, 令和 2 年 3 月 17 日.

ジコンピューティングの計算能力を約 1.36 倍高速化すると、VGG16 の 2 番目の pooling 層までエッジデバイス側で処理可能となり、推論処理にかかる消費エネルギーは、サーバーですべてのプロセスを実行する場合と比較して約 6.74 %削減できます。

キーワード

深層ニューラルネットワーク、エッジコンピューティング、推論、分散ネットワーク、ビデオ圧縮

Contents

List of Figures

List of Tables

1. Introduction

1.1 Background

Recently, machine learning has been progressed and integrated into many applications, such as object detection and pattern matching. These applications are used in a wide variety of situations. Automatic operation is an example. Deep Convolutional Neural Network (DCNN) achieved better recognition accuracy than that of human capability (5.1%)[18].

Conventional image recognition techniques consist of a feature extractor and a recognizer as shown in Fig. 1. The feature extractor is carefully and dedicatedly designed by researchers and developers. The recognizer is implemented by a simple neural network, such as SVM[16] and KNN[5]. However, designing practical feature extractors is extremely difficult and that capability is very limited. When the situation of imaging conditions, such as the angle of objects, is slightly changed, their output is largely affected and recognition accuracy is unacceptably degraded. Therefore, these methods can neither get the stable feature value nor reach the human recognition capability.

Deep Convolutional Neural Network (DCNN) consists of 2 main functionalities, feature extractor, which is implemented by a combination of convolutional and pooling layers, and classification, which is implemented by fully-connected layers. DCNN achieved 10% or more improvement of the recognition accuracy than conventional techniques. However, DCNN commonly needs a large amount of calculation. Due to enormous computational costs, mobile devices may not meet their real-time requirements. Therefore, the calculation of DCNN is usually processed on servers, having a huge amount of computing resources.

Here are two famous architectures of DCNN: AlexNet[8] and VGG16[20]. Both are excellent Neural Network for object recognition. AlexNet is composed of 5 convolutional layers, 3 pooling layers, and 3 fully-connected layers. The VGG16, which is deeper than AlexNet, is composed of 13 convolutional layers, 5 pooling layers, and 3 fully-connected layers. Image recognition using DCNN is implemented as shown in Fig. 2 and processes as follows.

1. The sensors get the data, compress and send them to the server.

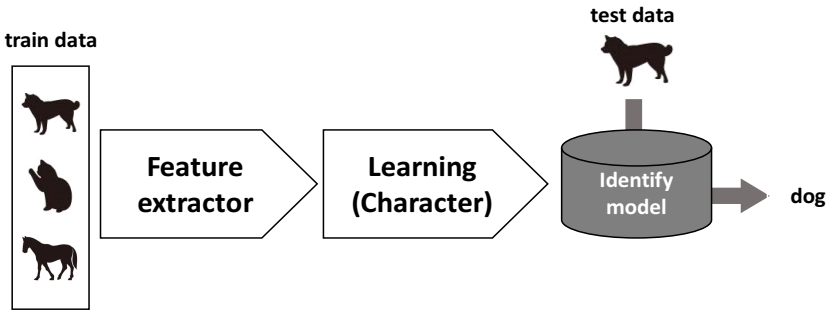

Figure 1: Conventional image recognition with SVM/KNN

2. The server receives and decompresses the data and executes the inference process.

3. The server sends the results back to the user.

However, recently proposed models of DCNN became much deeper and bigger than previous models. They require huge computation power. In the conventional method, all of DCNN are processed on servers, which have computing resources in abundance. As the size of DCNN grows and IoT devices increase[26], traffic congestion and the power consumption of network switches and servers are drastically increased. Due to this overconcentration, response latency becomes much longer. To cope with these issues, other computational infrastructures that are independent of the cloud server are required.

As a solution to these problems, many researchers and companies recently have paid high attention to the edge computing [7, 1] to process the data near their sources. So, I introduce that the process of a part of the network model could be executed by using edge computing. Our target is to implement the system to process machine learning in the dividable network model.

Since I need to find which optimal split boundary from the viewpoints of FPS, object recognition accuracy, and the energy consumption, I conducted a preliminary evaluation. By examining the values of intermediate data in the DCNN, I predicted which compression algorithm would be appropriate. Firstly, I confirm the results when applying HEVC compression to intermediate data of each layer. Then, I compared the compression ratio and recognition accuracy when using HEVC and lightweight compression. Third, I summarized the indicators

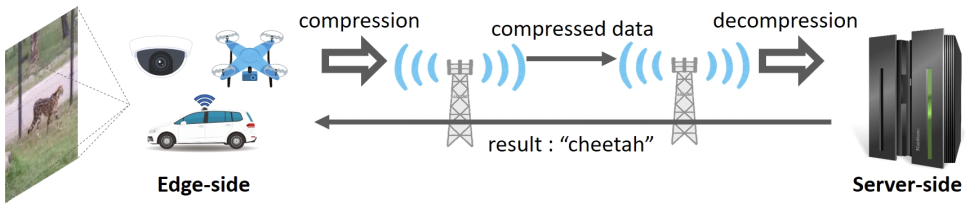

Figure 2: Inference processing in DCNN using servers

of the energy consumption in communication and inference process. Finally, in the comprehensive evaluation, I evaluate which split layer is optimal and whether compression is necessary.

1.2 Outline of Thesis

The remainder of this paper is organized as follows. Section 2 describes the preliminaries necessary to understand this study. Section 3 summarizes related work on approximation models, edge computing, and investigation for the sparsity of intermediate data. Section 4 summarizes the problems behind this study and the solution to problems. Section 5 shows preliminary evaluations to find the optimal split boundary and investigate for intermediate data. Section 6 indicates a method of applying compression to the intermediate data. Section 7 shows experimental setups and results in this study. A conclusion is described in Section 8.

2. Preliminaries

2.1 Neural Network

The origin of the Neural Networks is believed to be representing mathematically information processing in biological systems[11]. The single-layer Neural Network, expressed by Eq. (1), uses a linear sum of the input x and the weight parameter w as the input to the nonlinear activation function σ as one layer. In deep learning, it is called FCL (Fully Connected Layer). In learning phase for a multilayer Neural Network, it was difficult to proceed learning due to disappearance of gradient. Therefore, in many cases, a model of a little layer is proposed, and it is difficult to improve the accuracy as in a network with deep network.

$$y_j(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{\mathbf{i=0}}^{\mathbf{N}} \mathbf{w_{ji}x}\right) \tag{1}$$

2.1.1 Deep Neural Network

The reason that Deep Neural Networks have attracted attention is the result of the remarkable improvement in recognition accuracy at ILSVRC[18], a large-scale image recognition competition. In 2012, ILSVRC proposed AlexNet[8], a deep learning model with a total of 8 layers combining 5 CNNs and 3 FCLs. VGG is the network model that achieved second place in ILSVRC in 2014. VGG has more layers than AlexNet, so it has better information extraction capabilities. There are VGG16, which connects 13 CNNs and 3 FCLs, as a type of VGG.

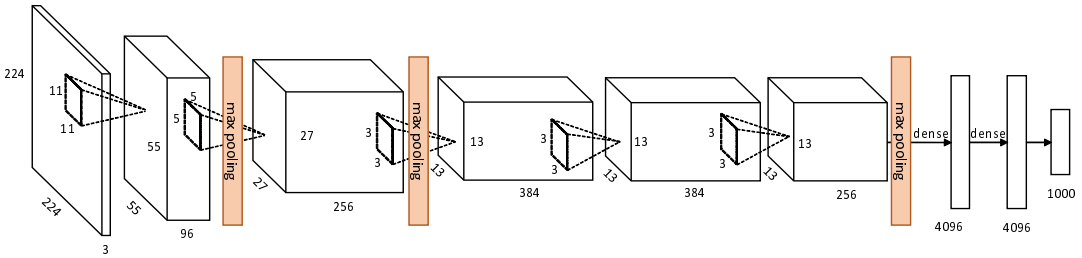

Figure 3: Structure of AlexNet

2.1.2 Convolutional Neural Network

In a DNN that takes an image as an input, the input to the fully connected layer that plays the role of recognition requires the feature data in the image. For this reason, a Neural Network for extracting feature values is provided at the preceding stage. The neural network that extracts the features of the image is called a Convolutional Neural Network (CNN). As shown in Fig. 4, when an input is given, the CNN performs an operation to obtain a sum of respective multiplications value obtained by extracting a window of the same size as the kernel from the weight parameters of the kernel of each convolutional layer. In short, one output is obtained from the convolutional process, and it obtains the feature data from applying this convolutional process to all inputs, the feature data by the convolution layer process is obtained. The output feature data passes through several convolutional layers to extract three-dimensional feature data from the image. Some feature data has a small amount of information and when all feature values are propagated, convolution operation and pattern recognition are performed on all of them. However, features with small values have little effect on the output of subsequent layers, and it is inefficient to perform calculations on all features. Several pooling layers with the roles of propagating only representative values from the output features are inserted between CNNs.

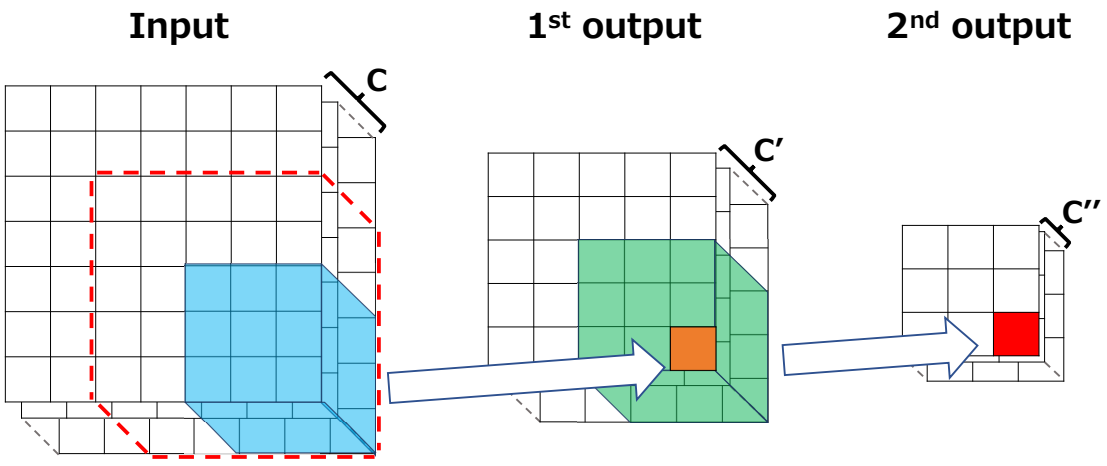

Figure 4: Convolutional Neural Network of two-layer

2.1.3 Pooling

Since the pooling layer can propagate only the representative value by passing through a filter having a certain size, the input to the subsequent Neural Network can be reduced. Max Pooling, proposed in the study of Zhou et al[27], shows a process of extracting only the maximum value in a square kernel for a feature map. For example, using a 2×2 two-dimensional kernel in Fig. 5, the number of elements in the two-dimensional feature map is reduced to one fourth. There are various types of pooling process such as "max pooling" and "average pooling". In application to object recognition, the max pooling is generally practical. By applying max pooling, there are advantages of being robust against small position changes and suppressing overfitting.

2.1.4 Activation Function

The activation function has a role in determining how the calculation result of the input signals is activated. This serves to simplify the values passed to the next layer. In general, the activation function of the "single perceptron" uses a step function, and the activation function of the "multilayer perceptron (Neural Network)" includes sigmoid and softmax function and identity function. Neural Networks can be used for both classification and regression problems. It is necessary to change the activation function of the output layer depending on whether it resolves a classification problem or a regression problem. Generally, an identity function is used in a regression problem, and a softmax function is used in a classification problem. The softmax function is represented by the following

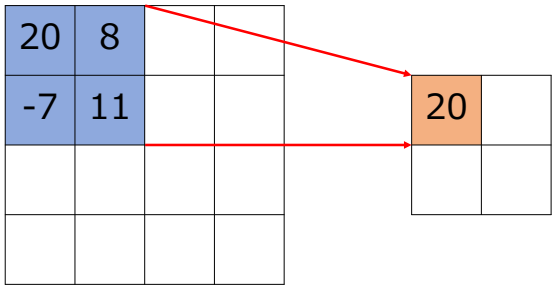

Figure 5: Max Pooling with 2×2 kernel

Eq. (2). n is the number of the output layer. y_k is the output value of k-th. Used between layers in the model such as AlexNet and VGG16, the function is the ReLU function. The ReLU function is given by following Eq. (3).

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)} \tag{2}$$

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{3}$$

2.2 Object Recognition

The opportunity to show the usefulness of deep learning widely was the large scale competition of algorithm for object recognition (ILSVRC). Starting with AlexNet, proposed by Hinton et al. in 2012, the application of deep learning to object recognition and other fields use deep learning models and learning methods that have shown the best accuracy every year. In the object recognition task, one class is classified for one image. When using a dataset with many classes and many images with high similarity between classes such as the 1000-class classification, I evaluate the accuracy with which the correct answer label is included in the top 5 classes of the classification results.

2.3 Compression

2.3.1 Lightweight Compression

Bzip2 Bzip2[19] is a lossless compression algorithm that combines the block sort, MTF (Move-to-Front), and Huffman coding to achieve efficient compression. Bzip2 achieves a higher compression ratio than conventional data encoding methods such as gzip and ZIP.

Run-Length Encoding Run-Length Encoding is a lossless compression algorithm alike bzip2. Run-Length Encoding expresses certain continuous data with a continuous length information and one character data. For example, Information with little color change, such as a monochrome bitmap image, is efficiently

Table 1: Differences between H.264/AVC and H.265/HEVC

	H.264/AVC	H.265/HEVC
Standardized time	2003	2013
Intra Prediction	16×16 ˜8×8	64×64 ˜8×8
Inter Prediction	9 mode	35 mode
Accuracy of Motion Vector	1/4 Pixel	1/4 Pixel
Orthogonal transformation	Integer precision 8×8, 4×4	Integer precision 32×32, 4×4
Loop filter	Deblocking filter	Deblocking filter, SAO (Sample Adaptive Offset)
Lossless compression	CAVLC, CABAC	CABAC

encoded, but is not suitable for encoding information of various types, such as text files.

2.3.2 H.265/HEVC

H.265/HEVC achieves about twice the compression ratio compared to the major compression technology H.264/MPEG-4 AVC. Table 1 shows the main technical differences between H.264/AVC and H.265/HEVC. Figure 6 shows a series of process of H.265/HEVC compression. HEVC was developed based on H.264/ AVC while improving individual elemental technologies.

In HEVC, the encoding process of each picture is executed in units of square pixel blocks called CTU (Coding Tree Unit). A series of encoding processes such as intra/inter picture prediction, DCT/quantization, and entropy encoding are executed in units of CU (Coding Unit) obtained by recursively dividing the CTU into quadtrees. For example, in areas where complicated motions and pictures exist, the CU size makes smaller, and the prediction performance is improved by allocating many codes to prediction parameters such as motion vectors and intra-picture prediction. Contrarily, for areas with flat patterns and uniform motion, by increasing the CU size and controlling the code amount of prediction parameters, it is possible to achieve high image quality even with a smaller code amount

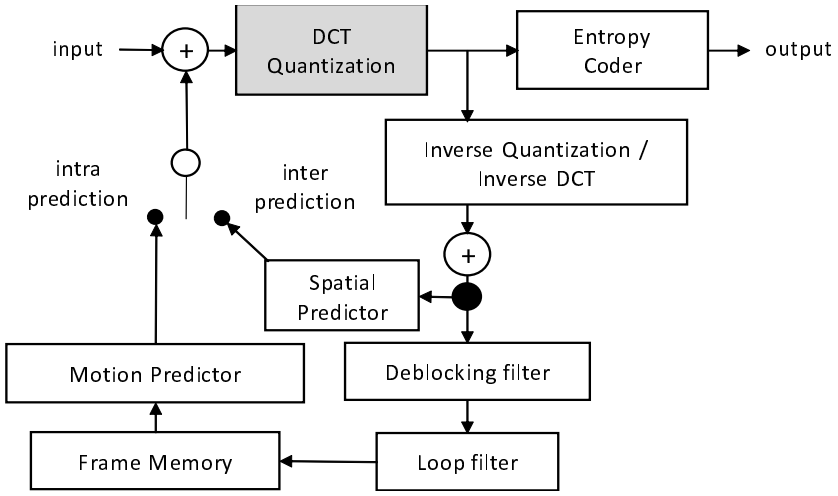

Figure 6: A series of processing of compression by H.265/HEVC

than conventional method. On the other hand, the computational complexity of the encoding process increased due to the increase in CU size variations and prediction parameter options. Therefore, in order to encode a high-resolution video in real time, it is necessary to perform processing using encoder. In the quantization processing of the black portion in Fig. 6, higher compression can be realized by changing the value of Qp (Quantization parameter) which is an option. A quantization without dead zone is used in which the logarithm of the quantization parameter and the quantization step (Qstep) is proportional (Qstep doubles when Qp increases by 6). The dead zone is the range of quantized values where the quantized value is 0. Enlarging deadzone increases the quantization error, but decreases the number of bits. The quantization process for getting the quantization coefficient q_{ij} of the orthogonal transform coefficient c_{ij} corresponding to the frequency component (i, j) is shown by the following Eqs. (4), (5) and (6).

$$qij = Int[cij/Qstep] \tag{4}$$

$$Qstep = \frac{m_{ij} \cdot 2^{qbit}}{Qscale(Qp\%6)} \tag{5}$$

$$Qbit = 25 + \frac{Qp}{6} - BitDepth - \log_2 N \tag{6}$$

Table 2: Quantization step coefficient

Qp%6	0	1	2	3	4	5
Qscale(Qp%6)	26,214	23,302	20,560	19,306	16,384	14564

$Int[]$ is function converting input to integer. m_{ij} is a quantization weighting factor. *BitDepth* is the bit length of the information to be processed. N is the size of orthogonal transformations. Also, the value of Qscale shows Table 2.

3. Related Work

3.1 Approximate Computing for CNNs

As Neural Networks evolve, a recognition based on CNN needs a large amount of memory and computational power. For that reason, while they work well on expensive machines, including high-end GPU, they are often unsuitable for smaller devices like cell phones and embedded devices. Recently, the construction of a neural network with reduced calculation cost and an equivalent model scale has been developed. Lightweight CNNs reduced precision computation have been developed. In Binary-Weight-Networks[10, 17], all the weight values are approximated with binary values. A convolutional neural network with binary weights is smaller ($1/32\times$) than the same network with single-precision floating-point weight values. Since weight values are binary, convolutions can be performed by only addition and subtraction (without multiplication), resulting in more than $2\times$ speed up. Also, XNOR-Nets[17] approximates both the weights and the inputs in the convolutional and fully connected layers to binary values. Binary weights and binary inputs allow an efficient calculation of convolutional operations. Since all of the operands of the convolutions are binary, the convolutions can be performed by XNOR and Bitcounting operations[10].

Ternary Weight Networks (TWNs)[9] is model compression method that has neural networks with weights constrained to $+1$, 0 and -1. TWNs have a more precise expression than the Binary-Weight-Networks. TWNs achieved higher accuracy than that of the Binary-Weight-Network. However, it is still lower than that of the full precision models. Like XNOR-Nets, there is a Neural Network model replacing floating-point matrix multiplications with ternary convolutions (based on sparse binary kernels), with both activation and weight restricted to values of $\{-1, 0, +1\}$[6]. This model can be calculated using a masked Hamming distance, an XOR/XNOR operation followed by a popcount, and reduce the computational cost by up to 1/16. This approach does not require high computational performance, but needs exploring the theoretical advantages of explicit sparsity promotion to reduce the risk of overfitting. To solve these problems, I propose a faster, more accurate and more efficient dividable model without excessive precision reduction.

3.2 Edge Computing

Edge computing[25] has been recently paid high attention. The reason is that it is important to increase power consumption due to increasing in transferring data and to construct an inference environment that reduces delays of transferring data. If most processes of DCNN are executed within the edge device and transfer data is smaller than sensor data, traffic congestion and over concentration in the cloud server are mitigated.

Edge devices can generally execute with lower power consumption than servers. Therefore, process of DCNN on the edge computing is more efficient than that on the server. Recently, Jetson TX2[15] developed by NVIDIA is a good candidate for the edge computing. Jetson TX2 is a modular AI supercomputer with NVIDIA Pascal architecture. This can execute the inference process with low energy consumption. Fig. 7 shows a photograph of the Jetson TX2 developer kit.

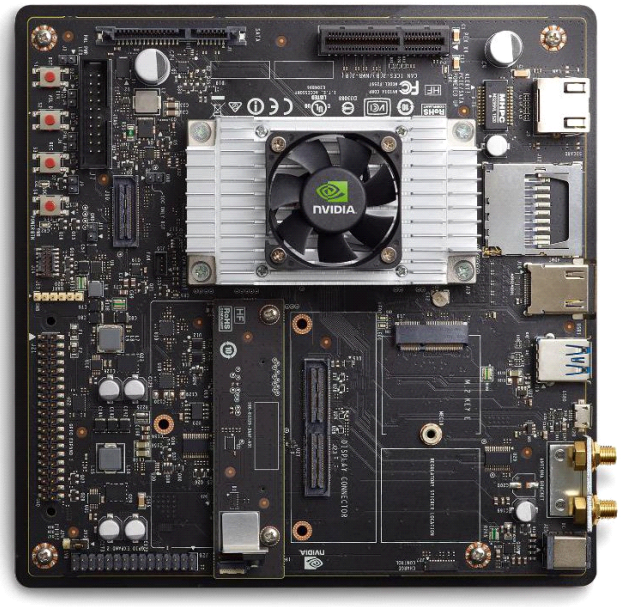

Figure 7: Jetson TX2 developer kit[15]

3.3 Efficient Compression Considering Sparsity

I mention the research that studies feature maps that are output in the latter layer of the DNN and applies an efficient compression to the feature map.

Reference[12] targets to develop a virtualization solution for DNNs that simultaneously meets the dual requirements of memory-scalability and high performance. If the processing time required to copy data to GPU memory via PCIe is longer than the computation of DNN forward and backward propagation process, there is overhead. It also says that data transferred to PCIe have high sparsity (the ratio converted to zero by Relu function) and high compressibility. The intermediate data output in the latter layer of AlexNet and VGG16 used in my study is found to be highly sparse. As a solution to this bottleneck, the paper proposes a compressing DMA engine (cDMA) for GPUs that eliminates the bottleneck of PCIe by reducing the size of the data structure copied around the GPU memory. Zero-value compression is one of the compression algorithms used in cDMA. Fig. 8 provides the overview of our zero-value compression (ZVC) algorithm which assumes a compression window sized as 32-bit consecutive data. For every 32 activation values, a 32-bit mask is generated with a '0' in a given bit position indicating the value is zero and a '1' indicating a non-zero value. After this 32-bit mask is generated, the non-zero elements are appended.

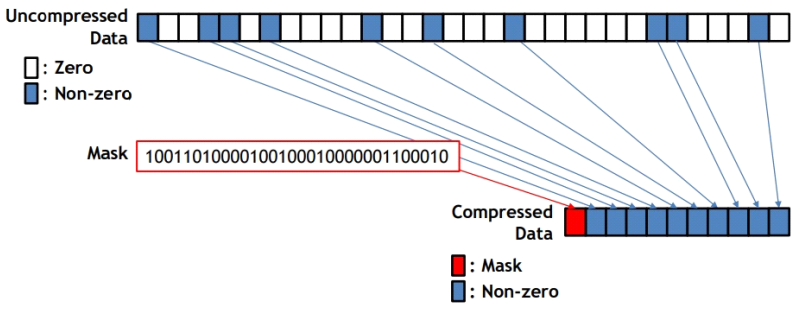

Figure 8: Algorithm of Zero-Value Compression[12]

4. Problem Definition

This section summarizes the reasons for assuming a divided model and conducting the preliminary evaluation in Section 2. I mention below future problems that will arise.

Increase in IoT Devices With the advancement of Internet technology and various sensor technologies, the IoT era has come. Various things around the world, such as home appliances, cars, buildings and factories, as well as PCs and smartphones, are connected to the Internet. There will be around 400 million IoT devices with cellular connections at the end of 2016 and that number is projected to reach 1.5 billion in 2022[3].

Increase in Communication Traffic With the sophistication of networks and the rapid increase of IoT, as you can see in Fig. 9, the amount of data traffic flowing through networks has increased dramatically. Total mobile data traffic continues to increase globally and is predicted to reach 131 Exabytes [EB] per month by the end of 2024[4]. Smartphones continue to generate most of the mobile data traffic. It is close to 90% today and projected to reach 95% at the end of 2024.

Increase in Energy Consumption Figure 10 shows the world energy consumption as measured by the IEA[2]. I found that there is a large difference in the ratio of increase in energy consumption between non-OECD and OECD countries. In OECD countries, growth in energy consumption is slower as a result of relatively slower population and economic growth, improvements in energy efficiency, and less growth in energy-intensive industries. However, non-OECD countries increases nearly 15% between 2018 and 2050. It still needs improvement to be efficient energy consumption.

From these problems, I thought that it is necessary to construct a divided network model that performs inference processing efficiently. By taking advantage of edge computing, it is possible to reduce the overall energy consumption by

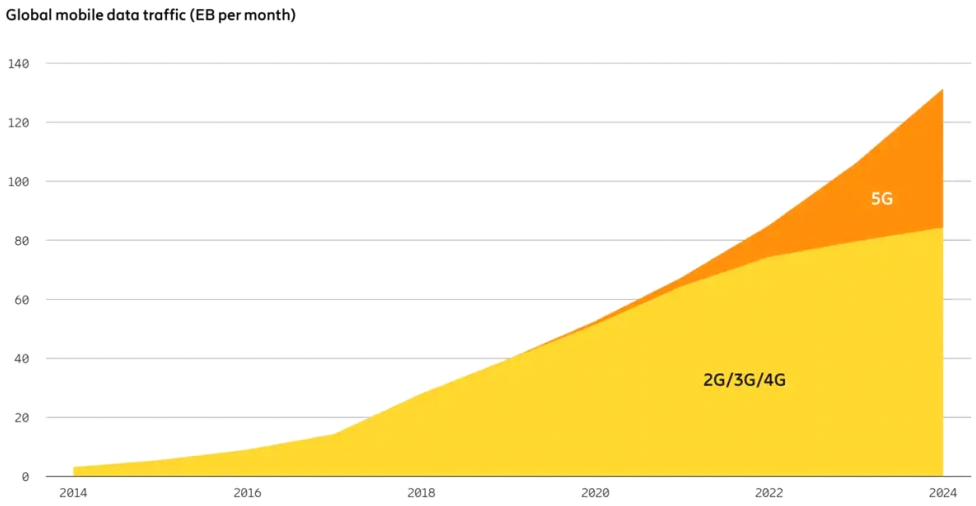

Figure 9: Global mobile data traffic [EB per month][4]

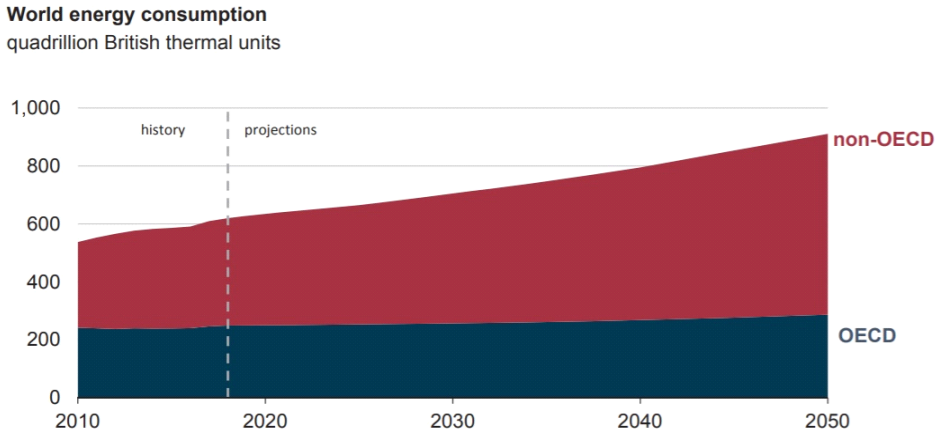

Figure 10: World energy consumption [quad][2]

performing the processing of the former layer on the edge-side. Hence, I need to explore at which split boundary to use.

Recently, 5G communication has become a hot topic. By implementing 5G communication, there are various advantages such as high speed, large capacity, low delay, and improvement in reliability. Accordingly, I am wondering if

advanced compression techniques like HEVC are needed to transfer intermediate data having sparsity. To solve this question, I need to compare HEVC with lightweight compression.

To summarize the discussion, I describe the following evaluation is necessary.

- Investigation in the amount of calculation and a volume of the intermediate data in the network model.
- Exploration of the characteristics of the intermediate data output from split boundaries and suitable compression
- Creation a performance and energy model to find a configuration that can inference process in real-time with the lowest energy

Section 5 conducts a preliminary evaluation of the above items.

5. Preliminary Evaluation

5.1 Amount of Output Data and Calculation in DCNNs

Figure 2 shows the inference process of DCNN on the server. Conventional implementation causes problems of network congestion and overconcentration of the calculation load on the server as described in Section 1. I split the layers of DCNN into two and assign them to multiple devices. That is to say, I introduce a dividable network model, which calculates DCNN more efficiently.

I explain the flow of the process of dividable DCNN using edge computing. When DCNN is implemented on two devices, edge device processes data from input data to a certain layer in DCNN, and the other device processes the remained layers. Figures 11 and 12 show that the volumes of output of each layer in AlexNet and VGG16. *conv*, *pool*, and *fc* correspond to convolution, max pooling, and fully-connected layers, respectively. When the input size is 224×224 and input bit-depth is 8 for each RGB color.

The bar charts in Figs. 13 and 14 represent the amount of computation for each layer, and the line charts represent the cumulative sum ratio from input to each layer. From Figs. 13 and 14, the total amount of computation is 2,777 [M operations] and 15,484 [M operations], respectively. That is to say, the calculation amount of VGG16 is 5.6 times larger than that of AlexNet. The convolution layer requires a larger amount of calculation than the other layers. This causes the overconcentration of computation. In order to balance the amount of computation, the split boundary should be between the convolutional layers. For example, if I set the split boundary to conv3_3 and conv4_1, the computation costs can be divided by 6:4. Moreover, since the amount of output data is reduced in the pooling layer, which is following the convolutional layer, the split boundary is desirable after the pooling layer.

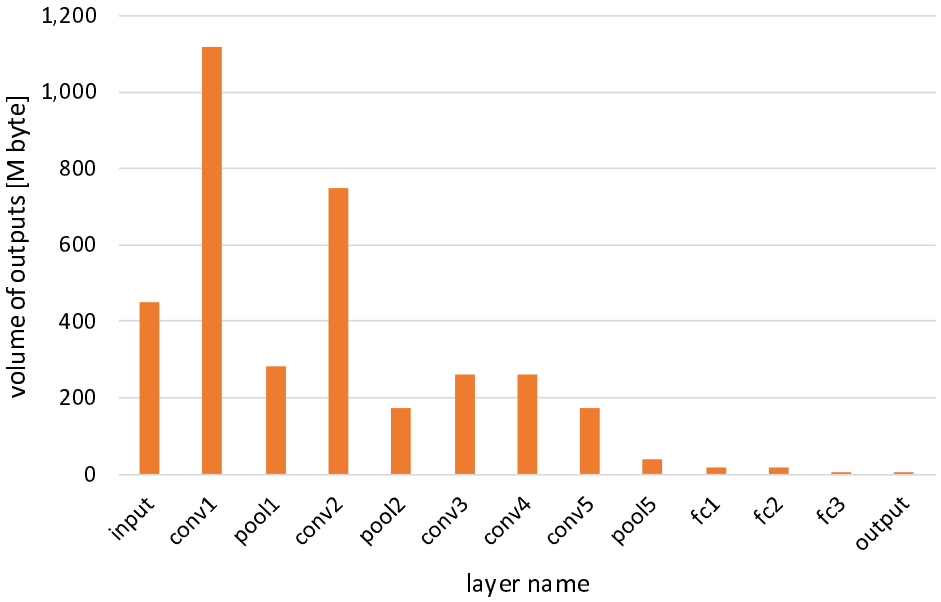

Figure 11: The volume of outputs on each layer in AlexNet

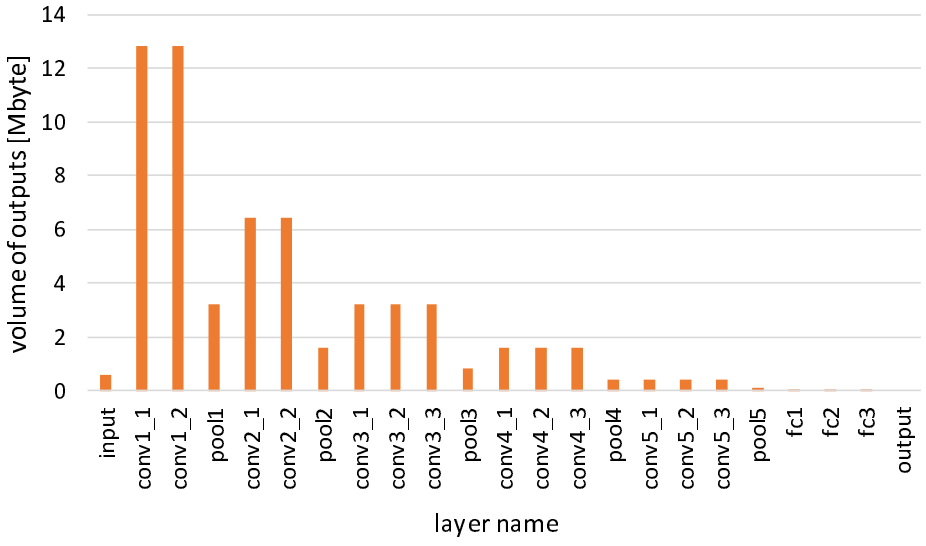

Figure 12: The volume of outputs on each layer in VGG16

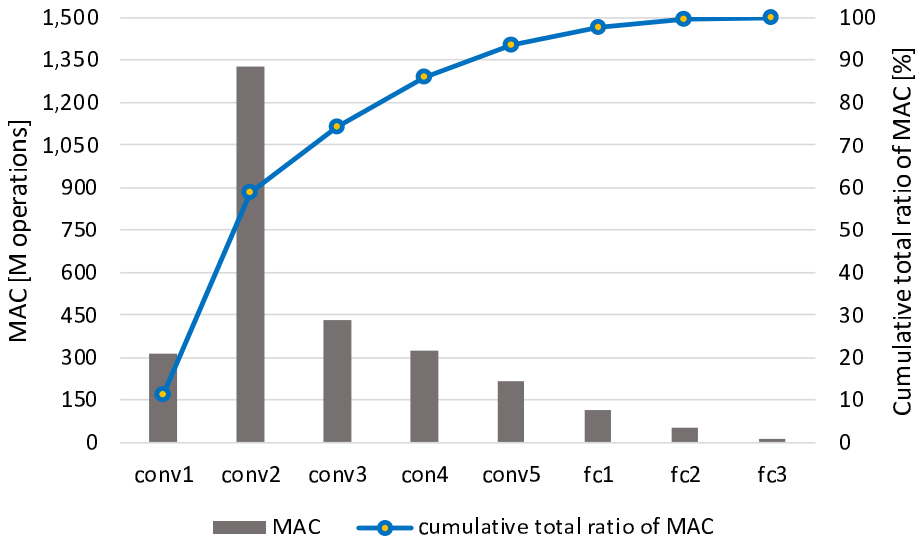

Figure 13: Amounts of calculations on each layer in AlexNet

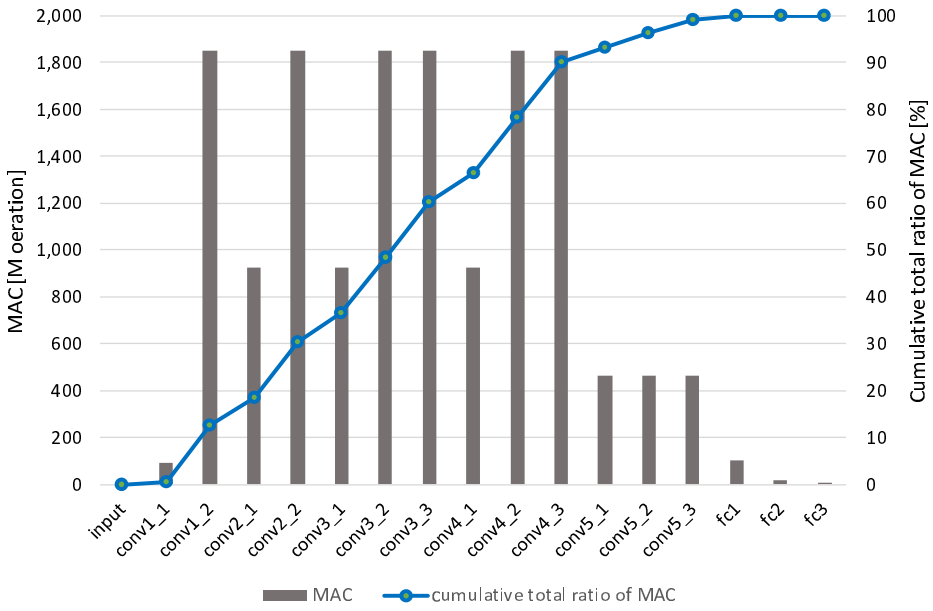

Figure 14: Amounts of calculations on each layer in VGG16

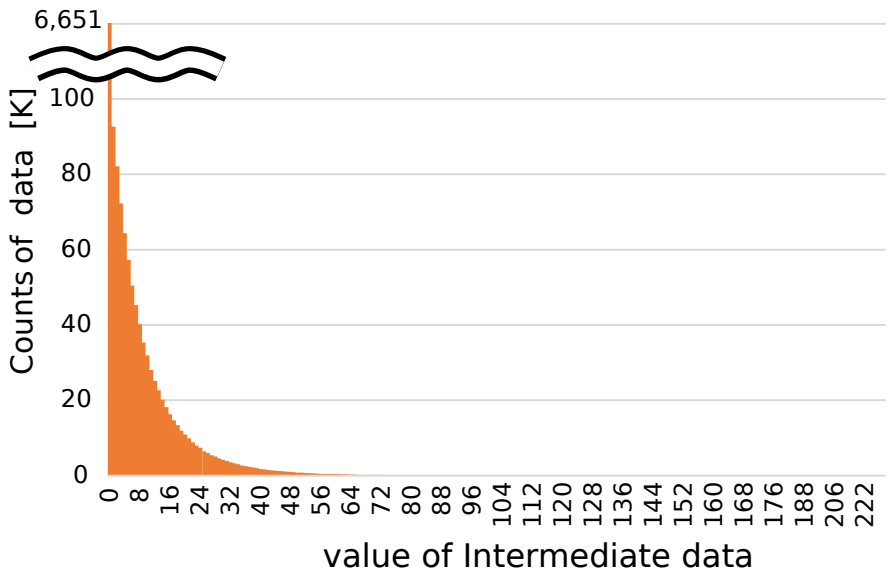

Figure 15: The value of output data from 5th pooling layer in VGG16

5.2 Investigation for the Intermediate Data in DCNN

For the intermediate data having sparsity, and the contents of the intermediate data were investigated in order to find lightweight compression realizing the compression ratio like HEVC and low power consumption. Figure 15 shows a frequency distribution of values of output data from the 5th pooling layer in VGG16. The output data is quantized from a float type to an 8-bit integer type for efficient compression. As test data, I used 300 images in ILSVRC2017. From this result, I found that most of the feature data is composed of zero values, and from 1 to 255 decreases exponentially. I need to consider a compression method suitable for this output feature map. Lightweight compression candidates include conventional methods such as Bzip2 and Run-Length Encoding, as well as zero-value compression introduced in Section 3.3. I did a comparative experiment of complex HEVC and its lightweight compression.

6. Proposed Method

6.1 H.265/HEVC for the Intermediate Data

For practical applications of image recognition in the IoT era, a popular type of input data is images that are continuously captured (i.e., Video) rather than independent images. In the technical competitions, huge numbers and a wide variety of pictures are suitable for measuring the average recognition accuracy. However, this kind of dataset is too far from reality. In this study, for practical uses, I treat the video stream obtained from surveillance or mobile communication cameras.

Since DCNN commonly requires a huge amount of computation, it is hard for a single device on the edge device to process them. At the same time, the divided approach requires network communication between devices. This overhead is not negligible.

The conventional implementation of DCNN requires data transfer from the sensors to the server. If the size of data is large and the communication costs are not negligible, the total performance is decreased. I can see that the output size of the input layer is smaller than the output size from pool1 in Fig. 11 and from pool4 in Fig. 12. I focus on this observation. Additionally, since the amount of data is still large, we compress the intermediate data of DCNN as well.

In previous research [13], H.265/HEVC compression[23] was adopted as a compression method for intermediate data of the network model. I thought that HEVC is optimal in this experiment because it achieves a large video compression ratio. HEVC compression has a minimum CTU size, which is 16×16. The window size of an output data from the pooling layer in the latter half of AlexNet or VGG16 is smaller than CTU size, so it cannot be compressed directly. Therefore, I need to compose multiple channel images by tiling as shown in Fig. 16 in order to fit the input of the HEVC compression.

The higher the recognition accuracy, the higher the reliability for the user as an actual application. In the previous study, only AlexNet was targeted, but in this experiment, VGG16 with a large number of layers is also used. By implementing a dividable network model, the overall energy consumption is expected to be lower than with conventional way of processing the entire inference process on

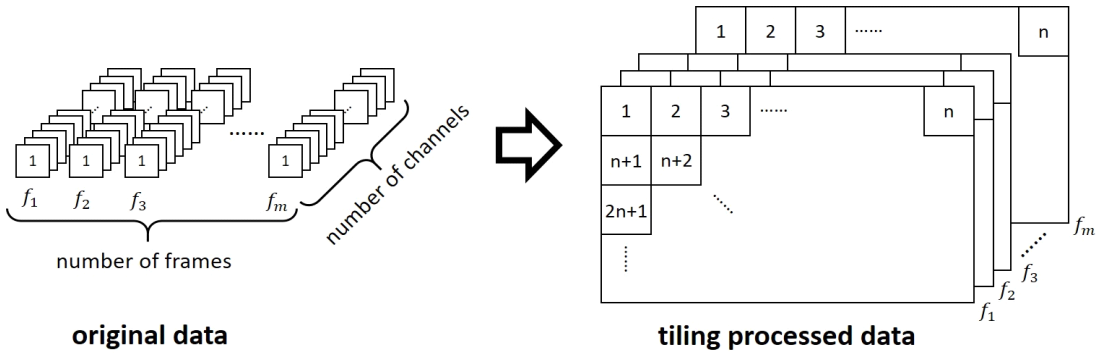

Figure 16: Frame composition from multi-channel image

the server.

6.2 Tolerant-RLE for the Intermediate Data

In the 5th pooling layer of VGG16, 90% of the values were zero. The Run-Length Encoding is suitable for compressing highly continuous information. Therefore, I investigated how changing the value of feature data affects the inference accuracy. In this experiment, the independent variable tol ($0 \leq tol \leq 255$) was introduced in the implementation. When the value of the output feature data is y, processing has been added to change the value of y to '0' if $y \leq tol$ is true. Thereby, the number of zero value increases, and the continuity of data also increases. Therefore, I expect the compression ratio increase using RLE.

7. Evaluation

Section 7.1 shows experimental setups for dividable DCNN. Section 7.2 summarizes experimental results.

7.1 Experimental Setups

7.1.1 Object Recognition

I investigate the efficiency of the video compression for the feature data extracted from the convolution layers of DCNN and the trade-off between the compression ratio and recognition accuracy. At first, Intermediate data of DCNN is extracted by the processing former layer. The first half of the model processed on the edge device. After compressing, transferring, and decompressing, the server continues the inference process and executes the object recognition task. In this experiment, I used 8 layers of AlexNet and 16 layers of VGG as the dividable network models. Since the pooling layer outputs the maximum value of the intermediate data through the 2×2 filter, pooling layer is desirable as a split boundary of each network model. AlexNet has three pooling layers and VGG16 has five pooling layers. Therefore, I respectively evaluate the object recognition accuracy and compressed data size when the split boundary is each pooling layer.

I used a Caffe Model Zoo[24] as the trained model of VGG16 and AlexNet in the experiment. This is the task of classifying 1000-class using the dataset published in the ILSVRC competition. As the validation data, I used the video of a cheetah walking from left to right. The video has 292 frames. I evaluated the accuracy when "cheetah" label was included in Top1 or Top5. Also, the details of the execution environment are detailed in Table 3.

7.1.2 Compression Methods

First of all, I describe the state of the intermediate data extracted from the former network model in the edge device. Table 4 shows the size of the intermediate data output in each pooling layer of AlexNet and VGG16. AlexNet targets three pooling layers, and VGG16 targets five pooling layers. The numbers in the Table 4 represent the size of the intermediate data, and represent (*channel*, *height*, *width*).

Table 3: Execution enviroment

Edge-side CPU	Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex
Server-side CPU	Core i7-3970X@3.90GHz(6 cores)
Edge-side GPU	Jetson TX2 (NVIDIA Pascal, 256 CUDA Cores)
Server-side GPU	GTX1080TiOC (NVIDIA Pascal, 3584 CUDA Cores)
Equipment for current measurment	SANWA Clamp meter DCM400AD
Equipment for voltage measurment	HP 972A Multimeter

Table 4: Size of intermediate data extracted from each pooling layer of AlexNet and VGG16

(*channel*, *height*, *width*)	AlexNet	VGG16
1th pooling	(96,27,27)	(64,112,112)
2th pooling	(256,13,13)	(128,56,56)
3th pooling	-	(256,28,28)
4th pooling	-	(512,14,14)
5th pooling	(256,6,6)	(512,7,7)

Second, I compared AlexNet with VGG16 constructed as dividable network models. For the dividable network models, I applied H.265/HEVC to intermediate data extracted from the former models. HEVC compression has a Quantization parameter (Qp) option. It manages the trade-off between compression ratio and image quality. HEVC has another parameter that specifies the GOP width of motion compensation (interframe prediction). The GOP is a group unit composed of multiple consecutive frames (I picture, P picture, B picture), and I picture is placed in the first frame. In this experiment, the GOP width is set to 30 frames (I:1, P:29, B:0). NvEnc encoder does not support creation for B pictures in many GPUs to realize real-time encoding. Therefore, the compressed

data is composed of I pictures and P pictures. In the dividable network models, I evaluated the compression ratio and object recognition accuracy. As noted in Section 6, HEVC compression has a minimum CTU size, which is 16×16. If the size of one frame is smaller than that, I cannot use HEVC. From Table 4, pooling *2* and *5* of AlexNet and pooling *4* and *5* of VGG16 correspond. Therefore, based on channel information, tiling process is required for these four intermediate data. However, in this experiment, when HEVC was applied, tiling processing was performed on all intermediate data.

Third, because of the complexity of the HEVC process, it is more common to use hardware encoders these days. Since NvEncoder and NvDecoder of the HEVC use GPU resources, I expect increases of power consumption. Therefore, I need to find a compression method that achieves the same compression ratio as HEVC and consumes low energy. I applied three different compression methods to the intermediate data, apart from HEVC. They are "bzip2" and "Run-Length Logarithmic Quantization Encoding" that are conventional methods, and, "Zero-Value Compression" suitable for compressing sparse data. "Run-Length Logarithmic Quantization Encoding" is a method of first performing logarithmic quantization of intermediate data, and then applying run-length encoding. Although the amount of information is largely lost, the compression ratio can be increased. For "Zero-Value Compression", the algorithm was described in Section 3.3, but in the related work, it has been applied to 32-bit data, but applied to 30 frames. I investigated the compression ratio, object recognition accuracy, and power consumption of compression when four different compression methods including HEVC were applied. Since "Run-Length Logarithmic Quantization Encoding" and "Zero-Value Compression" cannot achieve a high compression ratio unless the data is sparse, I applied it only to intermediate data extracted from 5th pooling layer in AlexNet and VGG16.

Finally, I applied "tolerant-RLE", the proposed method described in Section 6, to the intermediate data. The value of "tol" can be set between 0 and 255. Since the contents of information are changed, I expect that the recognition accuracy is greatly reduced depending on the test data.

7.1.3 Comprehensive Evaluation

Based on the above experiments, I evaluate the overall dividable network models. The following steps proceed step by step (1) extraction of intermediate data, (2) compression processing, (3) communication, (4) decompression processing, and (5) rest of the inference process. I assumed that the five processes of the dividable network models are performed by pipeline processing.

In this experiment, the HEVC compression/decompression processing is executed by a high-end GPU. NvEnc[14] is adopted when using HEVC. In the overall evaluation, I used all the processing time and the power consumption for the dividable network model. The energy consumption includes the total energy consumption per frame in the inference process on the edge device and the server, the compression/decompression processing, and communication.

In communication, I assumed Wi-Fi (40M[bps])[22], 3G (1M[bps])[22], and 5G (50M[bps]) as the effective communication speed of the channel. Our requirements are set to 30 FPS or higher for the processing speed and to 69.7% or higher achieved by XNOR-Nets, for object recognition accuracy.

7.1.4 Performance and Energy Models

The calculation method of FPS and energy consumption used in a comprehensive evaluation is described. In the inference process, t_{former} is the processing time executed on a GPU of the edge device, and t_{latter} is the processing time executed on a GPU of the server. The compression time is t_{comp} and the decompression time is t_{decomp}. I measured these values with real equipment. Also, the processing time required for communication is obtained by Eq. (7).

$$t_{comm} = size/throughput \tag{7}$$

Eq. (8) shows how to calculate FPS.

$$FPS = 1/\min(t_{former}, t_{comp}, t_{comm}, t_{decomp}, t_{latter}) \tag{8}$$

Finally, the measurement method of energy consumption is described below. The energy consumption J for the inference process and the compression/ decompression processing is calculated by Eq. (9),

$$J = P \cdot t \tag{9}$$

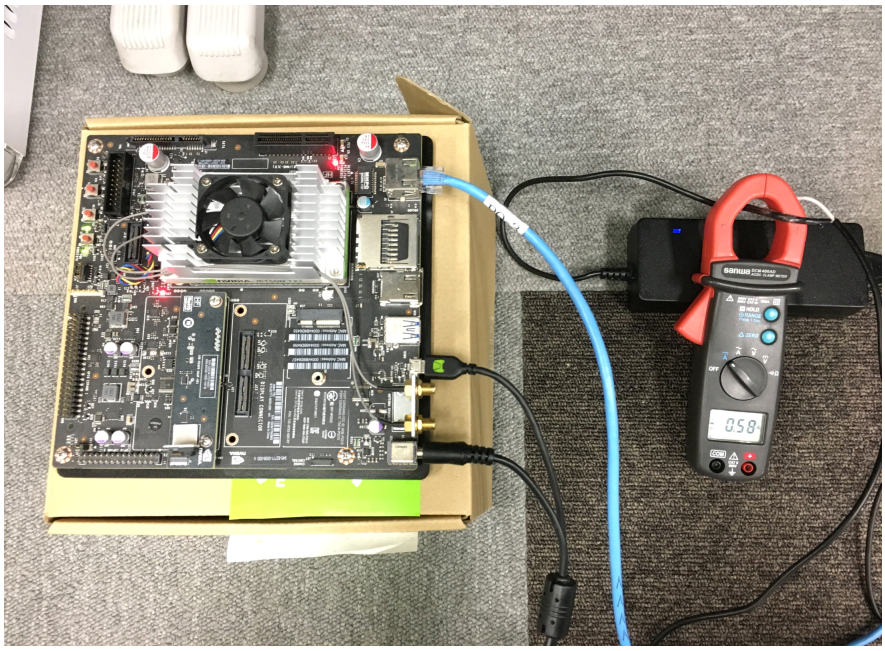

Figure 17: Measurement of current in the edge computing

which is a product of the power consumption (P) of the GPU and the processing time t. Table 3 shows the experimental equipment for the current and voltage measurement. The methods of measuring power consumption on the edge computing and the server are shown in Figs. 17 and 18. The current can be measured by passing the clamp meter through one end of the power cord. I used a multimeter to measure the voltage.

Table 5 summarizes the energy consumption for each communication standard in a J/bit. Multiplying the values in Table 5 and the size of the intermediate data are the energy consumption for communication. Since the HEVC compression tool in Jetson TX2 does not exist online, HEVC compression time is measured in GTX1080Ti. The power consumption is calculated by referring to the specifications of the commercially available chip for HEVC encoder[21].

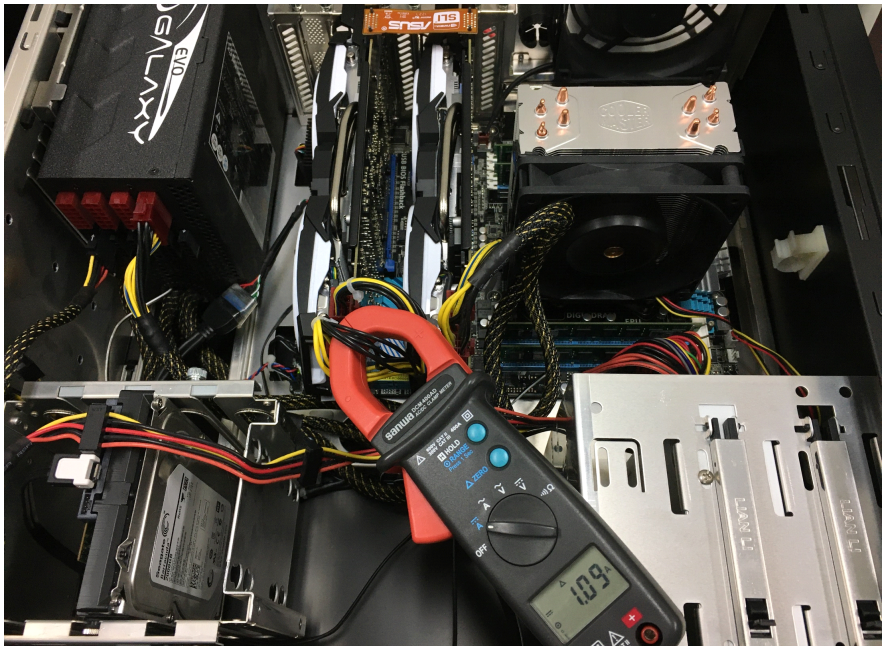

Figure 18: Measurement of current in the server

Table 5: Energy consumption in communication

Communication standard	Energy Consumption [J/bit]
Wi-Fi (40M [bps])	5.3×10^{-9}
3G (1M [bps])	3.5×10^{-7}
5G (50M [bps])	6.7×10^{-8}

7.2 Experimental Results

This chapter shows the result of experiments described in Section 7.1. Section 7.2.1 shows the result of applying HEVC. Section 7.2.2 shows a comparison between lightweight and video compression. Section 7.2.3 shows a result of applying tolerant-RLE. Section 7.2.4 shows measurement result of the power consumption used in the comprehensive evaluation. Section 7.2.5 summarizes the overall evaluation in terms of FPS, recognition accuracy, and energy consumption.

7.2.1 Divided Models using H.265/HEVC

Figure 19 shows the compressed data size and object recognition accuracy when HEVC is applied to the intermediate data output from both AlexNet [pool1, pool2, pool5] and VGG16 [pool1, pool2, pool3, pool4, pool5]. pool$h$ corresponds to the divided model when splitting at the h-th pooling layer of the model. The value of X-axis represents a Quantization parameter (Qp). This result clearly shows that the larger the Qp is, the greater the compression ratio of the amount of outputs in all layers will be. The solid lines of the chart represent the recognition accuracy of 1000-class in VGG16, and the broken lines represent the recognition accuracy of 1000-class in AlexNet. The bar charts represent the amount of data.

In both AlexNet and VGG16, the recognition accuracy is maintained up to $Qp = 25$, but the accuracy decreases sharply when $Qp = 30$ of pool2 in AlexNet. I found that the recognition accuracy was greatly reduced in the case of $Qp = 40$ and 50. When intermediate data are compressed with $Qp = 25$, compared to $Qp = 0$, in most cases, there is no difference in recognition accuracy, and the data size is greatly reduced. I found that the split boundary with the smallest data size of both AlexNet and VGG16 is pool5. This is because the original data is small. Compression ratio with Qp25 when the split boundary is pool5 in AlexNet and VGG16 has respectively realized data reduction of 1.37% and 0.57%. The recognition accuracy is maintained at 73.69% and 91.99%, respectively. Therefore, HEVC (Qp25) is adopted in the overall evaluation. From the viewpoint of the type of network model, I found VGG16 is the more robust divided model since AlexNet's accuracy changes greatly.

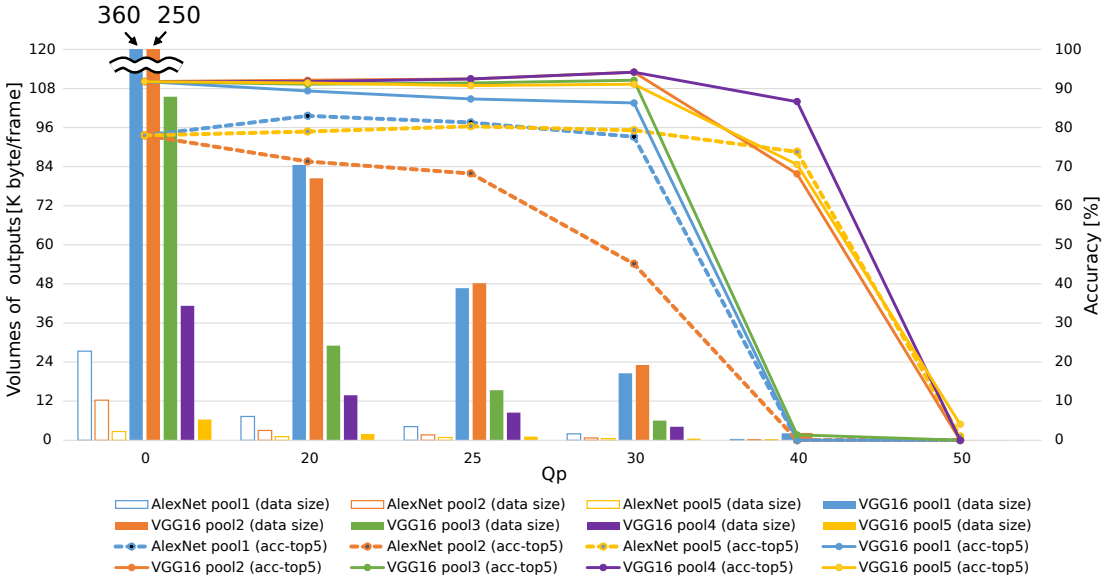

Figure 19: Volumes of output & accuracy of 1000-class applying H.265/HEVC to intermeidate data

7.2.2 Comparison between Lightweight and Video Compression

Figure 20 shows the result of applying four different compression methods to the intermediate data output from the 5th pooling layer in AlexNet and VGG16. The items on the horizontal axis indicate the applied compression method. Regarding HEVC, I used the case where $Qp = 0$, which has the least information loss, and the case where $Qp = 25$, which is just before the accuracy starts to decrease from Fig. 19. *rle* and *zero* indicates respectively "Run-Length Logarithmic Quantization Encoding" and "Zero-Value Compression". The bar graph shows the volumes of the compressed intermediate data per frame, and the line graph shows the top5 accuracy of 1000-class. From Fig. 20, I found that even with lightweight compression such as *rle-log*, *zero*, and *bzip2*, compression ratios as high as HEVC can be achieved for intermediate data having sparsity. In both AlexNet and VGG16, the compression method that realized the highest compression ratio is *rle-log*. In VGG16, compression ration of *HEVC (Qp25)* is 4.5%, while *rle-log* has successfully reduced the data size to 0.97%. In *HEVC (QP25)*, I found that the accuracy of the 1000-class was unstable because the value of the quantization

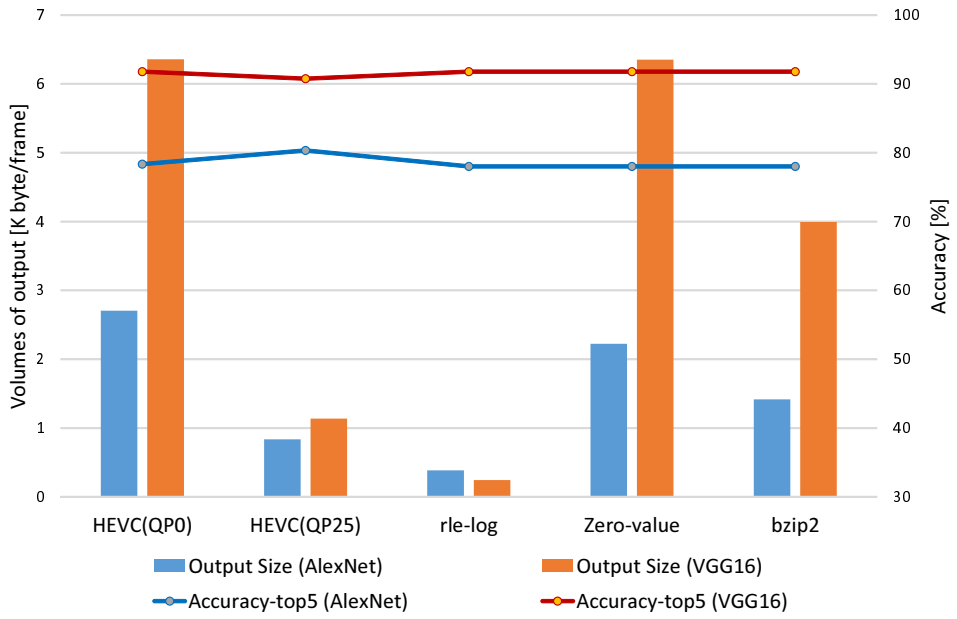

Figure 20: Volumes of output & accuracy of 1000-class applying four compression to intermeidate data

step was large and the amount of information was largely lost. I found that the accuracy of the other three compression methods in 8-bit quantization does not change and is practical.

Then, Table 6 shows the result of calculating the energy consumed by the compression processing. The items are the same as in Fig. 20. The numbers in the figure are the result of compression processing for each frame processed by edge computing. However, HEVC performed compression using GTX1080. The result shows that for both AlexNet and VGG16, *zero* is the least energy consuming. The intermediate data of 8-bit quantization of AlexNet is 2,691 [K byte], while VGG16 is 7,326 [K byte] which is about 2.7 times that of AlelxNet. The result of *rle-log*, *zero,* and *bzip2* also shows that the energy consumption in VGG16 is about twice that in AlexNet. In order to make a comprehensive evaluation, I must consider the transfer process.

Table 6: Energy consumption of four processing compression [mJ/frame]

	HEVC(Qp0)	HEVC(Qp25)	rle-log	zero	bzip2
AlexNet	24.1	14.5	9.04	**4.93**	7.40
VGG16	60.9	14.5	17.3	**10.7**	17.9

7.2.3 Tolerant-RLE

Figure 21 shows the result of applying "tolerant-RLE" to intermediate data output from *pool5* in AlexNet and VGG16. From this result, it is possible to investigate how the conversion of the value to '0' affects the recognition accuracy. The evaluation items are the same as in the previous section. The leftmost graph is the result of the *HEVC (Qp25)*. It is desirable that the data size is smaller than the data size of the HEVC and the recognition accuracy is maintained. As a result, in AlexNet, when $tol = 50$, the data size is smaller than *HEVC (Qp25)*. However, it turns out that accuracy decreased by 12.1%. That is below the target accuracy of 69.70%. In VGG16, I found that the data size is smaller than that and the accuracy was higher than the original accuracy by 0.69% when it is $tol = 35$. I found that changing the value of the intermediate data output from VGG16 between 0 and 60 had no effect. In order to investigate the reason for the difference in accuracy, the characteristics of the intermediate data between AlexNet and VGG16 are described. The numbers of intermediate data are 7,325K and 2,691K respectively. Also, the values of the standard deviation are 368K and 148K, respectively, and I found that the data values of AlexNet are discentral overall. Therefore, I see that the intermediate data of pool5 in AlexNet contain information that affects the latter stage, even at values close to zero.

7.2.4 Energy Consumption for processes of divided DCNNs

Table 7 shows the processing time and the energy consumption of the GPU in the edge device and the server used in AlexNet. Table 8 shows those of the GPU in the edge device and the server used in VGG16. I chose pool1, pool2, and pool5 as the split boundary for AlexNet, and chose from pool1 to pool5 for VGG16. "all-server" and "all-edge" of Tables 7 and 8 mean respectively execute the entire

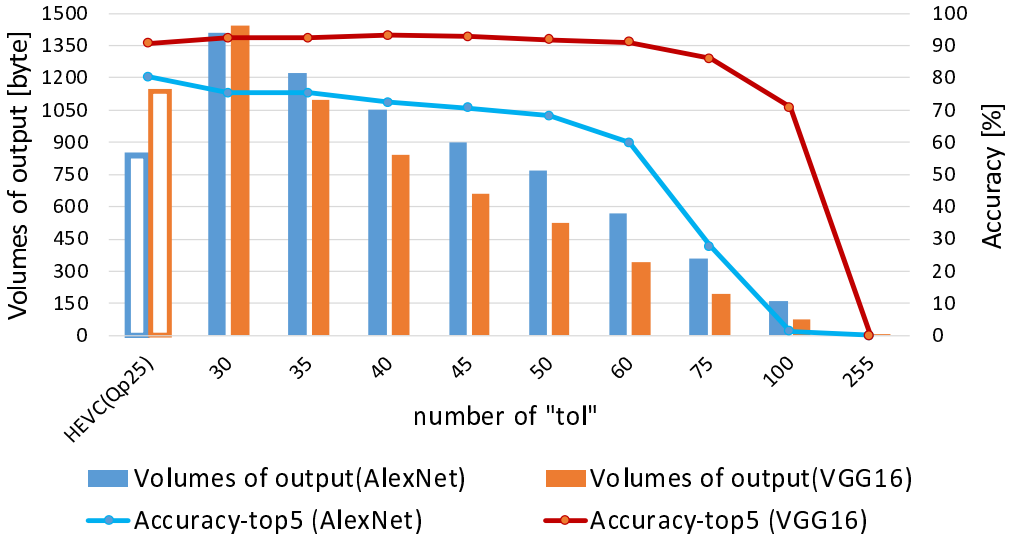

Figure 21: Volumes of output & accuracy of 1000-class applying "tolerant-RLE" to intermeidate data

Table 7: Inference processing time & energy consumption of the GPU for AlexNet

	Time [ms/frame]		Energy [mJ/frame]	
	Edge	Server	Edge	Server
all-server	-	5.37	-	152
1th pooling	8.70	2.32	50.1	71.4
2th pooling	9.06	2.12	69.6	57.3
5th pooling	9.25	0.681	74.6	15.9
all-edge	19.1	-	165	-

process on the server, execute the entire process on the edge device. I measured the energy consumption of the former and latter inference when executing at each split boundary.

After the former inference is over, the compression processing is applied to the intermediate data to reduce communication costs. Tables 9 and 10 show the result of processing time and energy consumption when the compression is applied to

Table 8: Inference processing time & energy consumption of GPU for VGG16

	Time [ms/frame]		Energy [mJ/frame]	
	Edge	Server	Edge	Server
all-server	-	7.05	-	347
1th pooling	32.3	5.52	186	272
2th pooling	45.3	4.27	331	200
3th pooling	66.0	3.39	507	150
4th pooling	84.8	2.50	651	98.2
5th pooling	90.1	2.10	692	64.5
all-edge	93.4	-	807	-

Table 9: Processing time & energy consumption with compression/decompression for feature data in AlexNet

	Time [ms/frame]		Energy [mJ/frame]	
	Comp	Decomp	Comp	Decomp
input	0.201	0.862	0.700	13.6
1th pooling	0.229	1.64	0.554	43.9
2th pooling	0.170	1.09	0.417	29.2
5th pooling	0.107	0.334	0.247	6.76

intermediate data for AlexNet and VGG16, respectively. "Comp" and "Decomp" indicate compression and decompression, respectively. In this experiment, HEVC is used for compression/decompression. The energy consumption of the 8-bit quantization process is so small that it can be negligible. "input" of these tables means a case in which HEVC is applied to sensor data.

For "all-edge," which execute the entire process on the edge device, the output is object IDs and the output size is only a few bytes. Therefore, both the processing time and the energy of compression/decompression and communication are negligibly short. In the overall evaluation shown later, these five tables are used to estimate the total energy consumption.

Table 10: Processing time & energy consumption with compression/ decompression for feature data in VGG16

	Time [ms/frame]		Energy [mJ/frame]	
	Comp	Decomp	Comp	Decomp
input	0.201	0.862	0.700	13.6
1th pooling	1.473	17.7	4.55	77.7
2th pooling	0.949	10.6	2.41	44.2
3th pooling	0.399	5.34	1.32	19.2
4th pooling	0.204	2.00	0.709	7.62
5th pooling	0.082	0.437	0.300	2.12

7.2.5 Comprehensive Evaluation

Based on the results obtained in Sects. 7.2.1 and 7.2.4, I performed a comprehensive evaluation. In Section 7.2.5, our requirements are 30 FPS or higher for the processing speed and 69.7%, which achieved XNOR-Nets, or higher for object recognition accuracy. Wi-Fi, 3G, and 5G communication standards were used in this experiment. Figures 22 and 23 show FPS and object recognition accuracy in dividable models of AlexNet and VGG16, respectively. Figure 24 shows the energy consumption in dividable models of AlexNet and VGG16. The red horizontal solid and broken lines in Figs. 22 and 23 represent the target value of 30 FPS and 69.7% accuracy achieved by XNOR-Nets, respectively. From left to right, the results of the entire process on the server, split boundaries (pool$h$), the entire process on the edge in AlexNet and VGG16 are shown. I compared the communication speed by Wi-Fi, 3G, and 5G, with their compression methods. "none (raw)" represents the data of 32bit-float type, 8-bit represents unsigned 8-bit quantization. hevc represents HEVC (Qp25). Zero-value compression and Logarithmic Quantization Encoding did not interfere with the conclusions of comprehensive evaluation. That is why they were omitted in the comprehensive evaluation. The "Energy" of Fig. 24 represents the total energy consumption in the inference process of the edge device and the server, the compression/decompression processing, and communication. The results of the energy consumption of pool3 and pool4,

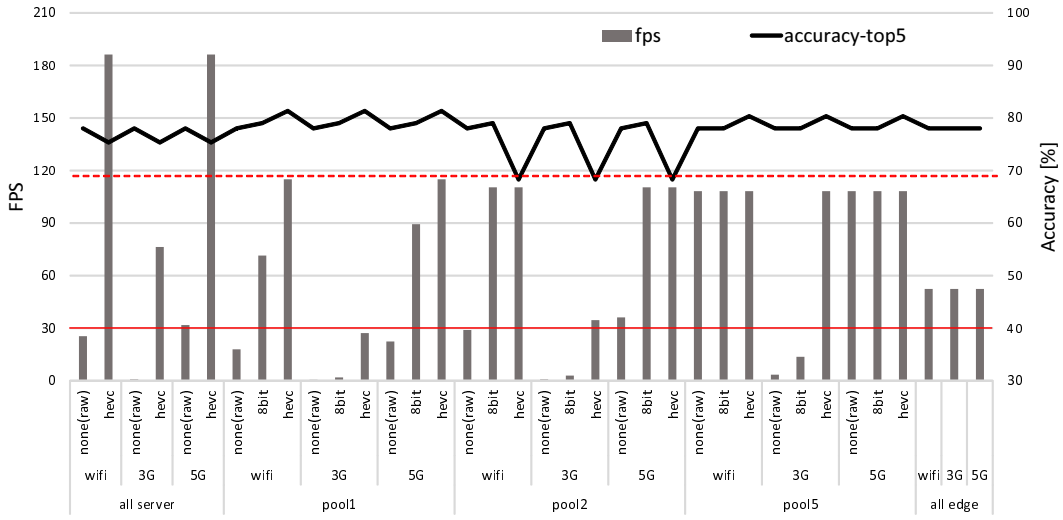

Figure 22: FPS & accuracy of 1000-class in divided AlexNet

which are the split boundaries of VGG16, are omitted. Because it does not meet the requirements from Fig. 23 and does not exist as a split boundary of AlexNet.

In Fig. 22, the object recognition accuracy of all combinations outputs a value of 69.7% or higher. These results clearly show that the requirement is satisfied. If the split boundary is pool5 and the communication environment is Wi-Fi or 5G, I found that this condition is satisfied even without HEVC compression and 8bit-quantization. In addition, all layers can satisfy 30 FPS by applying 8-bit quantization using Wi-Fi/5G. In the 3G environment, without HEVC compression cannot satisfy the conditions. Also, I found that "all-edge" of AlexNet can satisfy the conditions despite the low computational resources.

With Fig. 23, VGG16 has more layers and higher object recognition accuracy than AlexNet. Consequently, I found that the accuracy requirement is satisfied in all cases. I also found that the dividable network model in VGG16 could not satisfy 30 FPS when the process after pool1 is executed on the edge device. In pool1 with Wi-Fi or 5G, HEVC compression is required to satisfy 30 FPS. Also, In "all-server" with 5G, both none and HEVC satisfy the condition.

From Fig. 24, I found that under the same conditions, the energy consump-

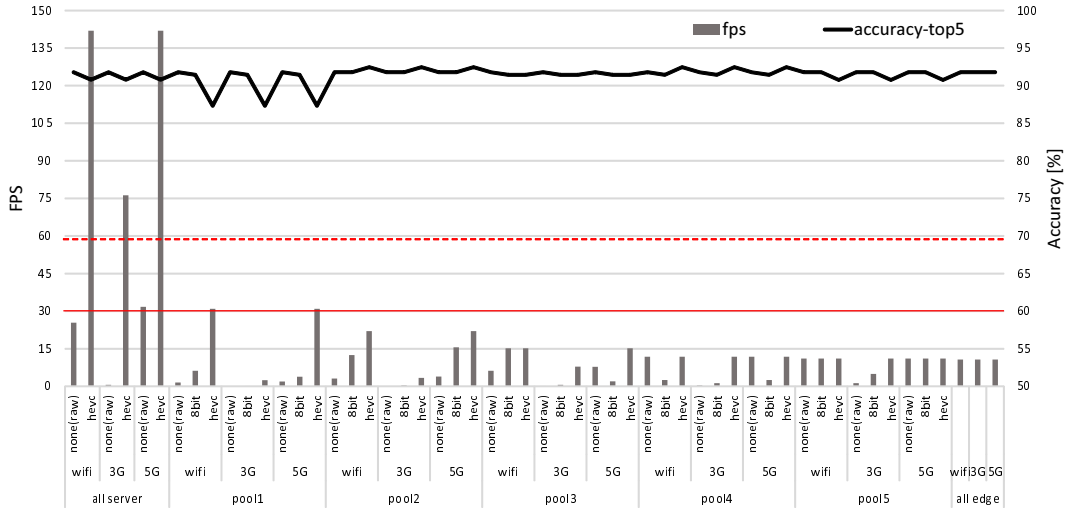

Figure 23: FPS & accuracy of 1000-class in divided VGG16

tion of AlexNet is smaller than that of VGG16, because the total computation of VGG16 is larger. In the case of 3G, although the power consumption for communication per bit is large and performance is low, requirements can be satisfied using HEVC reducing data size. On the other hand, since the value of J/bit is small in the Wi-Fi or 5G environment and the power consumption required for compression/decompression of the HEVC is larger, it is suitable to use only 8-bit quantization with Wi-Fi or 5G. Also, as you can see clearly, both AlexNet and VGG16 have the lowest energy consumption when the entire process is executed on the edge device.

Table 11 shows a summary of the result from the viewpoint of FPS, recognition accuracy, and the energy consumption per frame. *div (8bit)* and *div (hevc)* in the top row of Table 11 indicate that the network model is divided and 8-bit quantization or HEVC compression are applied respectively. "NG" indicates that the any combination does not meet some necessary conditions. *div (8bit)* and *div (hevc)* include all possible cases of the dividing boundary. The numbers represent the lowest energy consumption within all boundaries where the results of both FPS and accuracy conditions are met.

I expected to find an optimal split boundary in both AlexNet and VGG16 us-

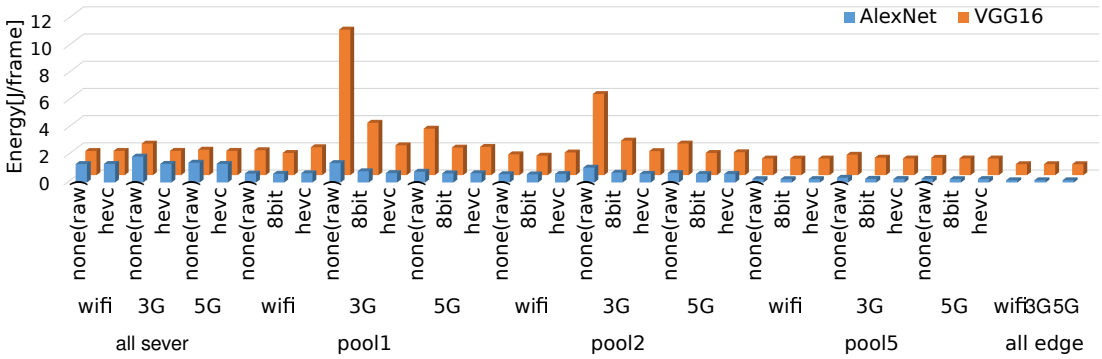

Figure 24: Energy consumption in divided AlexNet & VGG16

Table 11: Overall result for inference of divided AlexNet and VGG16 [J/frame]

		all-server (hevc)	div (8bit)	div (hevc)	all-edge (none)
AlexNet	Wi-Fi	1.36	0.246	0.253	**0.165**
	3G	1.37	NG	0.254	**0.165**
	5G	1.37	0.250	0.253	**0.165**
VGG16	Wi-Fi	**1.78**	NG	2.06	NG
	3G	**1.79**	NG	NG	NG
	5G	**1.78**	NG	2.08	NG

ing dividable network model. However, from this table, I found that in AlexNet, the least energy consumption is realized when the entire inference process is executed on the edge device in all cases of Wi-Fi/3G/5G. Energy consumption can be reduced by about 87.9% compared to executing all process in the server. Meanwhile, in the case of VGG16, if the split boundary is pool2 or later, FPS will be lower than the requirement due to long computation time on the edge device. If the split boundary is pool1, the amount of communication will be larger than input data. For this reason, the energy is greatly consumed for HEVC compression/decompression and communication. As a result, executing the entire inference process on the server with HEVC compression is optimal from a viewpoint of FPS, accuracy, and energy consumption. If the split boundary of VGG16

is pool2 or later and the communication environment is Wi-Fi or 5G, the power consumption is lower in most cases than "all-server". Therefore, when adopting VGG16, I found that it need measures to increase the processing capacity of edge computing or to somehow reduce the inference process of VGG16. If the computing power of edge computing is accelerated about 1.36 times, processing can be performed on the edge device up to pool2 in VGG16, and the energy consumption can be reduced by about 6.74% compared to the conventional method.

8. Conclusion

In this paper, regarding the network model with a large amount of computation, I considered dividable network models that can distribute the server load by using the computational resources on the edge computing. As a result of applying HEVC compression to the intermediate data, I found that the accuracy can be maintained by both AlexNet and VGG16 up to $Qp = 25$. When dividing in 5th pooling layer using HEVC, the compressed intermediate data can be reduced by 65.5% with AlexNet and 49.7% with VGG16 than the compressed sensor data.

However, the compression processing of the HEVC is complicated, and the energy consumption is large. Therefore, as a result of applying lightweight compression, I found that "zero-value compression" and "bzip2" consume less energy than HEVC for compression/decompression processing. However, since the compressed data size was about 5.9 times (zero-value compression) and about 3.5 times (bzip2) bigger than compressed data using HEVC (Qp25), I found that the power consumption for communication processing was large. In terms of "Run-Length Logarithmic Quantization Encoding" and "tolerant-RLE", depending on the type of data, it turned out that the compression ratio exceeded HEVC (Qp25) and kept the accuracy of 1000-class.

In order to construct models with efficient performance and energy, I evaluated the dividable model of AlexNet and VGG16. From the perspective of FPS, accuracy, and energy, the following has been found. When it is AlexNet, it is most efficient to execute all inference processing at the edge computing. If all inference processing is executed on the edge device, the output data is only 1000 normalized values, so compression processing will not be required. Therefore, regardless of whether the communication environment is Wi-Fi, 3G or 5G, the energy consumption is 0.165 [J/frame]. When it is VGG16, it is most efficient to execute all inference processing on the server using HEVC. Even with VGG16, I found that if the communication environment is changed, the energy consumption almost does not change. When Wi-Fi or 5G is set, 1.78 [J/frame]. When 3G is set, 1.79 [J/frame].

Finally, we investigated the dividable network model for object recognition in this study. As a result, an efficient implementation method of both network models was found. Regarding AlexNet, Energy consumption for inference can be

reduced by about 87.9% compared to executing all process in the server. Regarding VGG16, if the computing power of edge computing is accelerated about 1.36 times, processing can be performed on the edge device up to pool2 in VGG16, and the energy consumption for inference can be reduced by about 6.74% compared to executing all process in the server.

Acknowledgements

本研究に際して、ご多忙にも関わらず丁寧かつ熱心なご指導を頂き、また、研究の面だけでなく課外活動、日常生活まで多くのご指導をして頂きました本学の中島康彦教授に感謝の意を申し上げます。本稿をご精読いただき。発表の場においても貴重なご意見を頂いた本学の井上美智子教授に深く感謝致します。本研究へ多くの助言や、研究会の発表の場では勇気づけてくださったり、日常的に学生に近い目線で接して頂きました本学の中田尚准教授に感謝の意を表します。また日々の研究活動や本稿の執筆をはじめ多くのご助言を頂いた同研究室の TRAN Thi Hong 助教、張任遠助教に心より御礼申し上げます。日々の研究の進め方や研究室内の PBL やゼミナール、また就職活動に対しての助言等でご指導していただいた一倉孝宏氏、三谷剛正氏、上竹規之氏、平賀由利亜氏、菊谷雄真氏、山根弘樹氏、吉田怜矢氏、修士二年間を研究から日常生活まで支えてくれた龍谷大学の木村睦教授、同輩の池田裕哉氏、岩本淳氏、西本宏樹氏、後輩の本田卓氏、滝下雄太氏、野村武司氏、その他コンピューティング・アーキテクチャ研究室のメンバーと OB の方々からの助力に深く感謝します。

最後に大学院進学をするにあたり、経済的、心理的に今まで支えていただいた家族に感謝の意を申し上げます。

References

[1] Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[2] Ericsson. International Energy Outlook 2019. https://www.eia.gov/outlooks/ieo/pdf/ieo2019.pdf.

[3] Ericsson. Internet of Things forecast. https://www.ericsson.com/en/mobility-report/internet-of-things-forecast.

[4] Ericsson. Mobile data traffic outlook, 2019. https://www.ericsson.com/49d1d9/assets/local/mobility-report/documents/2019/ericsson-mobility-report-june-2019.pdf.

[5] Gongde Guo, Hui Wang, David A. Bell, Yaxin Bi, and Kieran Greer. KNN Model-Based Approach in Classification. In *CoopIS / DOA / ODBASE*, 2003.

[6] Mattias Heinrich, Maximilian Blendowski, and Ozan Oktay. TernaryNet: Faster Deep Model Inference without GPUs for Medical 3D Segmentation using Sparse and Binary Convolutions. *International Journal of Computer Assisted Radiology and Surgery*, 13, 01 2018.

[7] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing-a key technology towards 5g. *ETSI White Paper*, 11(11):1–16, 2015.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[9] Fengfu Li and Bin Liu. Ternary Weight Networks. *ArXiv*, abs/1605.04711, 2016.

[10] Courbariaux Matthieu, Bengio Yoshua, and David Jean-Pierre. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Advances in Neural Information Processing Systems - Volume 28*, NIPS'15, pages 3123–3131. Curran Associates, Inc., 2015.

[11] Christopher M.Bishop. Pattern recognition and machine learning (information science and statistics). Springer, 2006.

[12] Minsoo Rhu and Mike O'Connor and Chtterjee and Jeff Pool and Youngeun Kwon and Stephen W.Keckler and Postech and Nvidia. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. In *2018 ieee international symposium on high performance computer architecture*, pages 78–91, 2018.

[13] Takamasa Mitani, Hisakazu Hukuoka, Yuria Hiraga Takesi Nakada, and Yasuhiko Nakashima. Compression and Aggregation for Optimizing Information Transmission in Distributed CNN. In *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pages 112–118, 2017.

[14] NVIDIA. NVENC – NVIDIA HARDWARE VIDEO ENCODER, 2014. `http://developer.download.nvidia.com/compute/nvenc/v4.0/NVENC_AppNote.pdf`.

[15] NVIDIA. JETSON TX2 High Performance AI at the Edge, 2017. `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/`.

[16] P. Jonathon Phillips. Support Vector Machines Applied to Face Recognition. *Advances in Neural Information Processing Systems*, 11, 12 2001.

[17] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Lecture Notes in Computer Science - Volume 9908*, ECCV'16, pages 525–542. Springer, Cham, 2016.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual

Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[19] Julian Seward. bzip2 and libbzip2. In *a program and library for data compression*, 2000.

[20] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*, 09 2014.

[21] Socionext. H.265(HEVC) 4K/60p Small & Low Power Codec SC2M50. `https://www.socionext.com/en/products/assp/h264h265/SC2M50/`.

[22] Aaron Striegel, Shu Liu, Xueheng Hu, and Lei Meng. LTE and WiFi: Experiences with Quality and Consumption. *Procedia Computer Science*, 34:418–425, 2014. The 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops.

[23] Gary J Sullivan, Jens Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[24] Berkeley Vision and Learning Center. Caffe Model Zoo. `http://caffe.berkeleyvision.org/model_zoo.html`.

[25] W. Shi and J. Cao and Q. Zhang and Y. Li and L. Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[26] Felix Wortmann and Kristina Flüchter. Internet of Things. *Business & Information Systems Engineering*, 57(3):221–224, Jun 2015.

[27] Y.T. Zhou and Rama Chellappa. Computation of optical flow using a neural network. pages 71 – 78 vol.2, 08 1988.

Publication List

1. 新谷隆太, 中田　尚, 中島康彦: “分散 CNN における通信効率化のための圧縮技術の比較検討”, 信学技報, vol.119, no.147, CPSY2019-36, pp.203-208, Jul. (2019)

2. Ryuta SHINGAI, Yuria HIRAGA, Hisakazu Fukuoka, Takamasa MITANI, Takashi NAKADA, Yasuhiko NAKASHIMA: “Construction of an Efficient Divided/Distributed Network Model using Edge Computing”, IEICE Transactions on Information and Systems, The Institute of the Electronics, Information and Communication Engineers, 2019. (Under 1st Review)