Performance Evaluation of Pipeline-Based Processing for the Caffe Deep Learning Framework*

Ayae ICHINOSE†††, Nonmember, Atsuko TAKEFUSA††, Member, Hidemoto NAKADA†††, Nonmember, and Masato OGUCHI†, Member

SUMMARY Many life-log analysis applications, which transfer data from cameras and sensors to a Cloud and analyze them in the Cloud, have been developed as the use of various sensors and Cloud computing technologies has spread. However, difficulties arise because of the limited network bandwidth between such sensors and the Cloud. In addition, sending raw sensor data to a Cloud may introduce privacy issues. Therefore, we propose a pipelined method for distributed deep learning processing between sensors and the Cloud to reduce the amount of data sent to the Cloud and protect the privacy of users. In this study, we measured the processing times and evaluated the performance of our method using two different datasets. In addition, we performed experiments using three types of machines with different performance characteristics on the client side and compared the processing times. The experimental results show that the accuracy of deep learning with coarse-grained data is comparable to that achieved with the default parameter settings, and the proposed distributed processing method has performance advantages in cases of insufficient network bandwidth between realistic sensors and a Cloud environment. In addition, it is confirmed that the process that most affects the overall processing time varies depending on the machine performance on the client side, and the most efficient distribution method similarly differs.

key words: deep learning, machine learning, distributed processing, cloud computing, life-log analysis

1. Introduction

The spread of various sensors and Cloud computing technologies has made it easy to acquire various types of life logs and to accumulate the associated data. As a result, many life-log analysis applications, which transfer data from cameras and sensors to a Cloud and analyze them in the Cloud, have been developed. Cameras with a server function, called network cameras, have become cheap and readily available for security services and the monitoring of pets and children from remote locations. There has been much research on the Cloud-based analysis of sensor data and efficient Cloud-based analysis processing. In such services, raw data from sensors, including cameras, are generally transferred to a Cloud and processed there. However, it is difficult to send a large amount of data because of the limited network bandwidth between such sensors and a Cloud. In addition, sending raw sensor data to a Cloud may introduce privacy issues. As a measure to keep privacy, image scrambling might be used, for example. However, it can not necessarily guarantee the accuracy of analysis. In addition, it requires an additional image recognition cost of execution for image scrambling. In our proposed method, the accuracy is guaranteed, and privacy is preserved without an additional execution cost.

Deep learning is a neural network technique that is widely used for the analysis of images or videos. Deep learning makes it possible to automatically extract features from data; consequently, it has attracted the attention of researchers seeking improvements in accuracy and speed. Several deep learning frameworks have been developed, such as Caffe [1], TensorFlow [2], and Chainer [3]. Caffe enables high-speed processing and provides trained network models. The preparation of suitable network definitions is one barrier hindering the application of deep learning processing, but it is possible to use the network models provided by Caffe to easily perform experiments.

We propose a method of distributed deep learning processing between sensors and a Cloud to reduce the amount of data sent to the Cloud and enable to protect the privacy of users by sending preprocessed data. We have developed this technique extending the Caffe deep learning framework. In this work, only the execution time of a prediction phase is measured in the cases of performance evaluation without re-training nor fine-tuning. In addition, in the accuracy evaluation part in Sect. 4.2, pre-training is performed and prediction is executed using different datasets from those used in the pre-training. We split a deep learning processing sequence based on a neural network and perform distributed processing between the client side and the Cloud side in a pipelined manner. In this manuscript, we compare the processing times for classification in the following three cases: the whole processing is performed on the Cloud side, the processing is distributed using the proposed method, and the whole processing is performed on the Cloud side.

We investigate the performance characteristics using three types of machines with different performance characteristics on the client side, under the assumption using small sensors placed in general homes, sensors in a sensor box
for city monitoring, as in the Array of Things project [4], or a server close to the sensor, in an approach known as edge computing [5] or the Cloud Computing paradigm to the edge of the network [6]. We also investigate reduction of the learning accuracy when the division point of pipeline-based distributed processing and its parameters are varied to reduce the amount of data transferred to the Cloud. The experimental results show that the accuracy of deep learning with coarse-grained data is comparable to that achieved with the default parameter settings, and the proposed distributed processing has performance advantages in cases of insufficient network bandwidth between realistic sensors and a Cloud environment. In addition, we can see that when a low-performance client machine is used on the client side, the overall processing time depends on the amount of processing performed on the client side; whereas when a high-performance client machine is used, the overall processing time depends on the communication time, that is, the amount of data transferred. Thus, we can see that the most efficient distribution method differs depending on the balance between these two contributing effects.

The remainder of the paper is organized as follows. An overview of deep learning is provided in Sect. 2. In Sect. 3, a pipeline-based method of distributed deep learning is proposed, in which a convolutional neural network is split into two components, the client side and the Cloud side. The experimental environment is described in Sect. 4. A Linux server machine with a GPGPU is used on the Cloud side, and three types of machines are tested on the client side: a server machine and two types of sensor nodes, namely, the Jetson and the Raspberry Pi. In Sect. 5, the experimental results are presented. The three cases with different types of client nodes are compared using the CIFAR-10 dataset, and cases with different levels of network bandwidth are compared using the ImageNet dataset. Related work is introduced in Sect. 6. Finally, concluding remarks are provided in Sect. 7.

2. Deep Learning

Deep learning refers to a machine learning scheme that relies on a neural network with a large number of intermediate layers. A neural network is an information system that imitates the structure of the human cerebral cortex. It is able to achieve more exact recognition results by extracting various characteristics of the data of interest, such as colors, shapes, and more general aspects, in its intermediate layers. Currently, this approach is widely used for the recognition of both images and sounds. Caffe (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework developed by the Berkeley Vision and Learning Center (BVLC). Caffe comprises a combination of modules with specific functions, such as convolution and pooling, and the operation of the entire system is determined through communication between/within individual modules. This approach can be easily expanded to new data formats and network layers. The core of Caffe is written in C++; consequently, it is possible to use various user-friendly image classification tools, such as the Jupyter Notebook implemented in Python, via the Caffe C++ API. In addition, Caffe is capable of high-speed processing because it is compatible with GPUs, and the trained network models provided in the Caffe package enable the easy execution of experiments.

Caffe constructs a network architecture called a convolutional neural network. Convolutional neural networks are mainly applied for image recognition; such a network usually consists of repeated pairs of a convolution layer and a pooling layer, which are used to perform the basic calculations for image processing, followed by a fully connected layer. The convolution layers are used to compute the dot products between the entries of a filter and an input image at arbitrary positions and thus to extract a characteristic gray structure, as represented by the filter, from the image. The pooling layers take square areas of an input image and obtain a single pixel value corresponding to each, using the pixel values contained therein. This process lowers the position sensitivity of the features extracted in the convolution layers.

Here, we describe the details of each type of layer used in the Caffe network.

- **Convolution**
  A convolution layer performs convolution calculations. Here, it is necessary to set the number of filters and the kernel size, which represents the height and width of each filter. Optionally, it is possible to set a stride, representing an interval on which the filters are to be repeatedly applied to the input; a padding value, representing a number of pixels to be added to the edge of the input; and a group value, representing the number of divisions in the channels.

- **Pooling**
  A pooling layer performs pooling calculations. It is necessary to set the kernel size and, if possible, to set the method of pooling, the padding, and the stride. Examples of pooling methods include max pooling, in which the maximum value is selected from a set of pixel values, and mean pooling, in which the average value is calculated.

- **Local Response Normalization (LRN)**
  A local response normalization layer performs a type of “lateral inhibition” by normalizing across local input regions.

- **Inner Product**
  An inner product layer treats an input as a simple vector and produces an output in the form of a single vector.

3. Distributed Deep Learning Framework

We propose a pipeline-based distribution method, splitting a sequence of neural network processes as shown in Fig. 1. This approach makes it possible to protect the privacy of users by sending feature values instead of raw data; in addition, it can reduce the amount of data transferred to a Cloud,
which is beneficial for low-bandwidth environments.

We modified Caffe to allow a convolutional neural network to split into two components, the client side and the Cloud side. The client side and the Cloud side perform independent processes in Caffe. A “Sink” is located at the end of the client-side network and corresponds to a termination point. A “Source” is a starting point of the Cloud-side network. Each pair of Sink and Source are connected by TCP/IP connections. The Sink receives data from the upstream layer, transfers those data to its corresponding Source and waits for an ACK packet from the Source. The Source receives the data from the Sink, sends an ACK packet and then forwards the data to the downstream layers. Note that the client-side processes and the Cloud-side processes perform computations concurrently in a pipelined manner.

Examples of configuration files for a Sink and a Source are shown in Figs. 2 and 3. The Sink configuration file specifies the host name and port number for the corresponding Source process on the Cloud-side process. The Source configuration file specifies the port number. Note that the Sink layer also defines the shape of the matrix. This matrix size is required for the construction of the following layer stack.

When a network is split, more than one link may be cut. In such a case, a Sink-Source pair should be established for each cut link. Each pair is identified by its port number.

4. Experimental Settings

In this section, we introduce the network models and the distribution methods for the datasets used in the experiments. We also investigate the learning accuracy when the division point of distributed processing and its parameters are varied to reduce the amount of data transferred to the Cloud.

4.1 Datasets and the Division Points for Distributed Processing

4.1.1 CIFAR-10 Dataset

CIFAR-10 is a dataset in which images with dimensions of 32 × 32 pixels are classified into 10 categories, and the network model for this dataset is provided by Caffe. The structure of the network model is shown in Fig. 4. The parameters defined in each layer are shown in Table 1. Caffe stores and communicates data in 4-dimensional arrays, each of which specifies the batch size, the number of channels and the two-dimensional image size. The channel parameters are consistent with the number of filters in the immediately preceding convolution layer. The batch size for test was chosen to be 100; therefore, with the parameters set to the default values, the amount of data is initially (100 × 3 × 32 × 32) bytes.

- Distribution 1

As shown in Fig. 5, we split the network between the pool1 layer and the norm1 layer in a splitting scheme we call Distribution 1. We can reduce the number of filters in the conv1 layer to reduce the amount of data transferred. The numbers of filters in the conv2 layer and the conv3 layer are set to 32 and 64, respectively, which are the default values, whereas the number of filters in the conv1 layer can be varied from 1 to 32. The correspondence between the number of filters and the amount of data transferred is shown in Table 2.
Table 2  Correspondence between the numbers of filters in the conv1 and conv2 layers and the amounts of data communicated in the experiments using CIFAR-10 (KB).

| filters | 1   | 4   | 8   | 12  | 16  | 20  | 24  | 28  | 32  |
|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| conv1   | 25.6| 102.4 | 204.8 | 307.2 | 409.6 | 512.0 | 614.4 | 716.8 | 819.2 |
| conv2   | 6.4 | 25.6 | 51.2 | 76.8 | 102.4 | 128.0 | 153.6 | 179.2 | 204.8 |

• Distribution 2
  As shown in Fig. 6, we split the network between the pool2 layer and the norm2 layer in a splitting scheme we call Distribution 2. We can reduce the number of filters in the conv2 layer to reduce the amount of data transferred. The numbers of filters in the conv1 layer and the conv3 layer are set to 32 and 64, respectively, which are the default values, whereas the number of filters in the conv2 layer can be varied from 1 to 32. The correspondence between the number of filters and the amount of data transferred is shown in Table 2.

4.1.2 ImageNet Dataset
The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a competition that evaluates algorithms for object detection and image classification at a large scale. The test dataset for this competition consists of 150,000 photographs, collected from Flickr and other search engines, hand-labeled with the presence or absence of 1000 object categories. The network model for this dataset is also provided by Caffe, as is the one for CIFAR-10, as shown in Fig. 7. The batch size for the test was chosen to be 256; therefore, with the parameters sets to the default values, the amount of data is initially \((256 \times 3 \times 227 \times 227)\) bytes, which is reduced to \((256 \times 96 \times 55 \times 55)\) bytes after the conv1 layer, in which the number of filters is 96 and the stride is 4. Then, after the pool1 layer, in which the stride is 2, the amount of data becomes \((256 \times 96 \times 27 \times 27)\) bytes, which is approximately one half the size of the raw data. In the same manner, the amount of data decreases to approximately one third of the initial value after the pool2 layer. Here, we describe the two distribution methods utilized in the experiments reported in this paper, considering the amount of data transferred during communication.

• Distribution 1
  We split the network between the pool1 layer and the norm1 layer in Distribution 1 (Fig. 8). The amount of data transferred from the client side to the Cloud side is approximately half the size of the raw data, and we can further reduce the amount of data transferred by changing the number of filters in the conv1 layer. Here, we consider 72 and 48 filters, which are three fourths and one half of the default value, respectively.

• Distribution 2
  We split the network between the pool2 layer and the norm2 layer in Distribution 2 (Fig. 9). The amount of data transferred between the client side and the Cloud side is approximately one third the size of the raw data, and we can further reduce the amount of data transferred by changing the number of filters in the conv2 layer. Here, we consider 192 and 128 filters, which are...
4.2 Learning Accuracies with a Varying Number of Filters

Reducing the amount of data transferred between the layers of the neural network may decrease the accuracy of recognition; therefore, we investigate the accuracies achieved when we change the number of filters to reduce the amount of data transferred.

For the experiments using CIFAR-10, the correspondence between the number of filters and the amount of data transferred is shown in Table 2. The experimental results for CIFAR-10 using Distribution 1 are shown in Fig. 10, and those for Distribution 2 are shown in Fig. 11. The horizontal axes represent the number of filters, and the vertical axes represent the accuracy of identification.

We can see that the accuracies converge when the number of filters is small, as shown in Fig. 10 and Fig. 11. Even when the number of filters in the conv1 layer is 4, the accuracy remains at 73.35%, which is comparable to the result for the default value, 78.11%, although the amount of data transferred in the case of 4 filters is only one third the size of the raw data. With 4 filters in the conv2 layer, the accuracy remains at 73.35%, although the amount of data transferred is only one twelfth the size of the raw data.

In the experiments using ImageNet, as with CIFAR-10, high accuracy can be maintained with small numbers of filters, as shown in Table 3 and Table 4.

Hence, we can see that high accuracy can be maintained even if we reduce the amount of data transferred by reducing the number of filters in the convolution layers.

5. Comparison of Processing Times

We demonstrate the effectiveness of the proposed method by reporting the processing times for the identification of 1 batch using two machines, one on the client side and one on the Cloud side. In this section, only the execution time of a prediction phase is measured without re-training nor fine-tuning. We compare the following three cases:

1. The whole processing is performed on the client side.
2. The processing is distributed between the client and Cloud sides using the proposed method.
3. The whole processing is performed on the cloud side.

In cases (2) and (3), the client side and the Cloud side run simultaneously, so we use the processing times measured from the client side, including connection times and waiting times. Before the experiment begins, TCP connection is already established so that the connection waiting time is not included in the evaluation results.

In Sect. 5.2, three cases with different types of client nodes are compared. On the Cloud side, a Linux server ma-
Table 5  Performance of the machine used on the Cloud side.

|       |                |
|-------|----------------|
| OS    | Ubuntu 16.04.1 LTS |
| CPU   | Intel(R) Xeon(R) CPU W5590 @3.33 GHz (4 cores) × 2 sockets |
| Memory| 8 Gbyte         |
| GPGPU | NVIDIA GeForce GTX 980 |

Table 6  Performance of the Jetson.

|       |                |
|-------|----------------|
| OS    | Ubuntu 14.04.5 LTS |
| CPU   | NVIDIA 4-Plus-1 Quad-Core ARM Cortex-A15 |
| Memory| 8 Gbyte         |
| GPU   | NVIDIA Tegra124 PM375 |

Table 7  Performance of the Raspberry Pi.

|       |                |
|-------|----------------|
| OS    | Linux raspberrypi 4.1.19+ |
| CPU   | 900 MHz Quad-Core ARM Cortex-A7r |
| Memory| 1 Gbyte         |

Machine with a GPGPU is used in all cases. On the client side, the following three types of nodes are used: a node of the same quality as the server machine but without a GPGPU, a high-performance sensor node (Jetson), and a normal sensor node (Raspberry Pi). These three cases are evaluated using the CIFAR-10 dataset.

In Sect. 5.3, cases with different levels of network bandwidth are compared. In this experiment, server machines of the same quality are used on both the Cloud and client sides, but a GPGPU is used only on the Cloud side. These cases are evaluated using the ImageNet dataset.

5.1 Machine Performance

The performance specifications of the machine used on the Cloud side in all experiments are listed in Table 5. In the experiments using CIFAR-10, in addition to experiments with same-quality nodes, we also tested the use of the Jetson and Raspberry Pi platforms on the client side. The performance specifications of the Jetson and the Raspberry Pi are listed in Table 6 and Table 7, respectively. In the experiments using ImageNet, we used nodes of the same quality on both the client side and the Cloud side; however, we used only the CPU on the client side and the GPGPU on the Cloud side. We used PSPacer [7] to control the network bandwidth to represent various sensor and Cloud network environments.

5.2 Experiments using Different Types of Client Nodes

We compare the processing times achieved using three types of client nodes. In all experiments, the network bandwidth between the client side and the Cloud side was set to 10 Mbps.

5.2.1 Case with Same-Quality Nodes on the Client and Cloud Sides

Figures 12 and 13 show the processing times of CIFAR-10 obtained for Distribution 1 and Distribution 2 with a client-side node of the same quality as the Cloud-side node but using only the CPU. The horizontal axes represent the number of filters in the immediately preceding convolutional layer, and the vertical axes represent the processing time. For Distribution 1, the results for case (2) are faster than those for case (3) when the number of filters is fewer than 8, that is, when the amount of data transferred is smaller than the size of the raw data. For Distribution 2, the results for case (2) are faster than those for case (3) when the number of filters is fewer than 28 because the amount of data transferred is smaller than the size of the raw data in all of these cases. We can see that the results predominantly depend on the communication time rather than the image processing time because of the use of a high-performance node on the client side; consequently, a reduction in the amount of data transferred to the Cloud greatly affects the total processing time. Although the results for case (1), in which the whole processing is performed on the client side without communication, are faster than those for the other cases, it is typically not possible to use such a high-performance node on the client side.

5.2.2 Case with a High-Performance Sensor Node (Jetson)

We performed experiments using a Jetson node, which is a somewhat high-performance sensor node, on the client side.
Figures 14 and 15 show the processing times for Distribution 1 and Distribution 2 when the Jetson’s GPU is used, and Figs. 16 and 17 show the corresponding results when only the CPU is used.

With the use of the GPU, the processing times for cases (1) and (2) are lower than in the case of same-quality nodes because the client-side processes are faster. The proportion of the communication time to the total processing time is further increased, and the results for case (2) are faster than those for case (3) when the number of filters is set to the default in the experiments with Distribution 2. In the experiments with Distribution 1, when the number of filters is set to the default value of 32, the amount of transferred data is higher, causing the processing time for case (2) to be approximately 2.7 times that for case (3). However, when the number of filters is set to 8 or fewer and the amount of transferred data is reduced, case (2) is faster than case (3).

When only the Jetson’s CPU is used, the processing on the client side is slower, so the image processing time on the client side also affects the overall processing time. Both the communication time and the amount of processing on the client side have a considerable influence on the overall processing time, and even in the experiments with Distribution 2, the times for case (2) are longer than those for the other cases. In addition, in the experiments with Distribution 1, the amount of processing on the client side is such that case (2) is faster than both other cases when the number of filters is set to 4, that is, the communication time is reduced. Unlike the case with a client-side node of the same quality as the Cloud-side node but using only the CPU, with a higher number of filters, case (3) is faster than both other cases for both Distribution 1 and Distribution 2. We can see that the overall processing times depend on the amount of processing performed on the client side because the processing on the client side is slow.

5.2.3 Case with a Normal Sensor Node (Raspberry Pi)

We performed experiments using a Raspberry Pi, which is regarded as an ordinary sensor node, on the client side. Figures 18 and 19 show the results for Distribution 1 and Distribution 2, respectively. Case (3) is faster than the other cases for both Distribution 1 and Distribution 2, as the experiments using only the Jetson’s CPU. The overall processing times depend on the amount of processing performed on the client side because the processing on the client side is slow.

5.2.4 Discussion

We compared experimental results obtained using machines
with different performance specifications on the client side, and the results confirmed that the process that most affects the overall processing time varies depending on the machine performance and that the most efficient distribution method similarly differs. When the performance of the client node is high, the overall processing time predominantly depends on the communication time between the client and the Cloud; consequently, in the case of the proposed method, which can reduce the amount of data transferred to the Cloud, the processing time can be reduced. When the performance of the client node is low, the amount of processing performed on the machine with lower performance greatly affects the overall processing time; consequently, the case in which the whole processing is performed on the Cloud side is faster than the other cases.

5.3 Experiments with Varying Levels of Network Bandwidth

We measured the processing times of ImageNet achieved with a fixed number of filters and a varying network bandwidth of 1 Gbps to 10 Mbps between the two machines. In this subsection, we experiment with a client side-node of the same quality as the Cloud-side node but using only the CPU.

Figures 20, 21 and 22 show the results for Distribution 1 with the number of filters in the conv1 layer set to 96, 72 and 48, respectively, and Figs. 23, 24 and 25 show the results for Distribution 2 with the number of filters in the conv2 layer set to 256, 192 and 128, respectively.

In both experiments, the results for case (3) are faster than those for the other cases when the network bandwidth between the two machines is 1 Gbps. However, for bandwidth levels of 10 Mbps and 50 Mbps, case (3) is slower than case (2) because the size of the raw data is large, resulting in a longer communication time, and consequently, the total processing time is longer. In addition, the processing time in the distributed processing case can be further reduced by reducing the number of filters. When the net-
work bandwidth between the two machines is 50 Mbps, the processing time for case (2) and Distribution 2 is faster than those for the other cases, and when the network bandwidth between the two machines is 50 Mbps or 100 Mbps, the case (2) result is also faster for Distribution 1. Thus, we can see that the most efficient distribution method depends on the network bandwidth and the performance of the machines on both the client side and the Cloud side.

5.4 Generalization of Performance Model

While the results of performance evaluation are shown in Sect. 5.2 and 5.3, they are only parts of cases in which conditions are different in a real environment. Therefore, generalization of performance model is discussed in this subsection.

The total processing time of case (1) Client side is almost equal to the volume of processed data divided by the power of processing at client side. We denote this value as $X$. The total processing time of case (3) Cloud side is almost equal to the volume of transferred data divided by the bandwidth of the network, if the processing power is higher enough so that the processing time at the Cloud is negligible compared with the transferred time. We denote this value as $Y$. In case (2) Distribution, the processing time at client side is reduced compared with that of case (1) Client side, because only the part of processing is executed at the client side. we can denote this value as $\alpha X$ ($0 < \alpha < 1$). The transfer time in this case is also reduced compared with that of case (3) Cloud side, because the volume of transferred data is reduced as the data are processed and filtered at the client side. We can denote this value as $\beta Y$ ($0 < \beta$).

In summary, the approximations of the total processing time in each case are as follows:
- Case (1) Client side: $X$
- Case (2) Distribution: $\alpha X + \beta Y$ ($0 < \alpha < 1, 0 < \beta$)
- Case (3) Cloud side: $Y$

Based on this simple performance model, the total processing time can be displayed as the function of $X$ and $Y$, as shown in Fig. 26. In this figure, the x axis is the normalized value of $X$ and the y axis is the normalized value of $Y$. As an example, $\alpha$ is set to 1/2 and $\beta$ is set to 1/3 in this figure, which is roughly the case of Fig. 23 shown in Sect. 5.3. In this example, the total processing time of case (2) Distribution is shorter than those of case (1) Client side and case (3) Cloud side when the following condition is satisfied:

$$3/4* X < Y < 3/2* X$$

The instance of bandwidth being 50M [bps] in Fig. 23 should be in this case as the total processing time of case (2) Distribution is the shortest.

Figure 27 shows the one with the minimum processing time among the three cases in this performance model when $\alpha$ and $\beta$ vary from 0.0 to 1.0. The vertical axis represents the value of $\alpha$ and the horizontal axis represents the value of $\beta$. The blue areas represent the case (1) Client side is the
fastest, the red areas represent the case (2) Distribution is the fastest, and the green areas represent the case (3) Cloud side is fastest. The condition that red areas exist is $\alpha + \beta < 1$ as shown in Fig. 27. This is because $\alpha X + \beta Y < X, Y$, therefore $(\beta/(1 - \alpha)) * Y < X < ((1 - \beta)/\alpha) * Y$. The condition that $X, Y$ exist is $\beta/(1 - \alpha) < (1 - \beta)/\alpha$, therefore $\alpha + \beta < 1$.

Actually, the values of $X$ and $Y$ are decided and cannot be changed easily in a real environment. As shown in Fig. 26, although the total processing time of case (1) Client side or case (3) Cloud side is shorter than that of case (2) Distribution depending on the condition, those cases are rather extreme, i.e., the power of processing at client side is very high or the bandwidth of network is very high in IoT environments in general. Compared with those cases, case (2) Distribution achieves the best performance in a condition that is close to relatively normal. Due to the limit of the experimental environment, only parts of the cases are shown in Sect. 5.2 and 5.3, and only some of them are the cases when case (2) Distribution achieves the best performance. However, when we see those results are the part of generalized model as shown in Fig. 26 as example, we can conclude that case (2) Distribution are effective in relatively wide variety of cases, compare with case (1) Client side and case (3) Cloud side cases which requires rather extreme conditions.

### 6. Related Work

Neural networks have been widely used to precisely identify and classify patterns [8], [9]. However, for many neural network frameworks, high-speed performance is achieved by utilizing the GPU in a single computer.

In [10], Caffe was executed in various Cloud environments. In this research, a fully compatible end-to-end version of the popular Caffe framework with rebuilt internals was developed. They achieved a $6.3 \times$ throughput improvement over Caffe for popular networks such as CaffeNet. With these improvements, the end-to-end training time for CNNs is directly proportional to the FLOPS delivered by the CPU, which allows hybrid CPU-GPU systems for CNNs to be effectively trained. However, this enables high-speed processing only in Cloud environments, not in distributed environments. In [11], a system for automatic font identification using a neural network in a client-server environment was developed. However, this research presented a system running on a thin client. In this system, only I/O processing is performed on the client side, and recognition using the neural network is performed on the server side. Xnor.ai [12] enables learning solely on client devices with low processing capacity, without the need to connect to a server.

Unlike in these works, in our research, load discrimination is performed with respect to identification processing performed by a neural network split between the client side and the Cloud side, thereby enabling more appropriate processing.

In [13], GraphLab framework which naturally expresses asynchronous, dynamic, graph-parallel computation is realized and achieving a high degree of parallel performance in distributed environment. In our research work, we performed sensor data analysis in distributed environment.

The DIANNE middleware framework is another related work [14]. Whereas a normal neural network consists of an input layer, an output layer and one or more hidden layers, in the DIANNE middleware framework, each layer of the neural network is composed of modules. This modular approach makes it possible to execute various components of a single neural network on a large number of heterogeneous devices. However, the amount of transferred data and the processing time for distributed computing are not considered in the cited work. By contrast, we strive to reduce the processing time by considering the amount of transferred data, and based on this effort, we propose an efficient framework for life-log analysis processing for general consumers.

### 7. Conclusions

We propose a pipeline-based distributed processing method for deep learning and implemented distributed processing for the Caffe deep learning framework with the aim of sensor data analysis, considering privacy issues and a limited network bandwidth. In the case of a realistic network bandwidth between general homes and a Cloud, non-negligible time is required to transfer raw sensor data, and thus, the proposed method is beneficial. In this study, we observed that it is possible to maintain a high accuracy and achieve efficient processing even when the amount of data transferred data a sensor to a Cloud is reduced by reducing the number of filters in one of the convolution layers.

In addition, we performed experiments using three types of machines with different performance specifications on the client side and compared the resulting processing times. We observed that when a low-performance machine is used on the client side, the overall processing time depends on the amount of processing performed on the client.
side, whereas in the case of a high-performance client-side machine, it depends on the communication time, that is, the amount of data transferred. Therefore, the most efficient distribution method differs depend on the balance between these two contributing effects.

In the future, we will perform further experiments using multiple nodes on the client side to more closely resemble a typical real environment.

Acknowledgments

This paper is partially based on results obtained through a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and supported by JSPS KAKENHI Grant Number JP16K00177.

References

[1] Y. Jia, et al., “Caffe: Convolutional architecture for fast feature embedding,” arXiv preprint arXiv:1408.5093, 2014.
[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” pp.1–19, 2015. http://download.tensorflow.org/paper:whitepaper2015.pdf
[3] S. Tokui, K. Oono, S. Hido, and J. Clayton, “Chainer: A next-generation open source framework for deep learning,” Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), 6 pages, 2015.
[4] “Array of things.” https://arrayofthings.github.io/
[5] P.G. Lopez, A. Montresor, D. Epena, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, “Edge-centric computing: Vision and challenges,” SIGCOMM Comput. Commun. Rev., vol.45, no.5, pp.37–42, Sept. 2015.
[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things.” Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC ’12, New York, NY, USA, pp.13–16, ACM, 2012.
[7] R. Takano, T. Kudoh, Y. Kodama, and F. Okazaki, “High-resolution timer-based packet pacing mechanism on linux operating system,” IEICE Transactions on Communications, vol.E94-B, no.8, pp.2199–2207, 2011.
[8] A. Krizhevsky, I. Sutskever, and G.E. Hinton, “Imagenet classification with deep convolutional neural networks,” Advances in Neural Information Processing Systems 25, ed. F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, pp.1097–1105, 2012.
[9] S. Pierre, E. David, Z. Xiang, M. Michaël, F. Rob, and L. Yann, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” Proceedings of International Conference on Learning Representations, 15 pages, 2013.
[10] S. Hadjis, F. Abuzaied, C. Zhang, and C. Ré, “Caffe con troll: Shallow ideas to speed up deep learning,” Proceedings of the Fourth Workshop on Data Analytics in the Cloud, DanaC’15, pp.2:1–2:4, 2015.
[11] Z. Wang, J. Yang, H. Jin, J. Brandt, E. Shechtman, A. Agarwala, Z. Wang, Y. Song, J. Hsieh, S. Kong, and T. Huang, “Deepfont: A system for font system and similarity,” Proceedings of the 23rd ACM international conference on Multimedia (MM’15), pp.813–814, Oct. 2015.
[12] “Xnor.ai.” https://xnor.ai/
[13] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J.M. Hellerstein, “Distributed graphlab: A framework for machine learning and data mining in the cloud,” Proc. VLDB Endow., vol.5, no.8, pp.716–727, April 2012.
[14] E. De Coninck, T. VerbeLEN, B. Vankeirsbilck, S. Bohez, S. Leroux, and P. Simoens, “Dianne: Distributed artificial neural networks for the internet of things,” Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT, M4IoT 2015, New York, NY, USA, pp.19–24, ACM, 2015.