Pipeline-Based Processing of the Deep Learning Framework Caffe

Ayae Ichinose
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku,
Tokyo, 112-8610, Japan
ayae@ogl.is.ocha.ac.jp

Atsuko Takefusa
National Institute of Informatics
2-1-2 Hitotsubashi,
Chiyoda-ku, Tokyo 101-8430, Japan
takefusa@nii.ac.jp

Hidemoto Nakada
National Institute of Advanced Industrial Science and Technology (AIST)
2-3-26 Aomi, Koto-ku, Tokyo 135-0064, Japan
hide-nakada@aist.go.jp

Masato Oguchi
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku,
Tokyo, 112-8610, Japan
oguchi@is.ocha.ac.jp

ABSTRACT

Many life-log analysis applications, which transfer data from cameras and sensors to a Cloud and analyze them in the Cloud, have been developed with the spread of various sensors and Cloud computing technologies. However, difficulties arise because of the limitation of the network bandwidth between the sensors and the Cloud. In addition, sending raw sensor data to a Cloud may introduce privacy issues.
Therefore, we propose distributed deep learning processing between sensors and the Cloud in a pipeline manner to reduce the amount of data sent to the Cloud and protect the privacy of the users. In this paper, we have developed a pipeline-based distributed processing method for the Caffe deep learning framework and investigated the processing times of the classification by varying a division point and the parameters of the network models using data sets, CIFAR-10 and ImageNet. The experiments show that the accuracy of deep learning with coarse-grain data is comparable to that with the default parameter settings, and the proposed distributed processing method has performance advantages in cases of insufficient network bandwidth with actual sensors and a Cloud environment.

CCS Concepts

•Computer systems organization → Client-server architectures; Neural networks; *Pipeline computing;* **•Computing methodologies** → Machine learning;

IMCOM '17 January 05–07, 2017, Beppu, Japan

DOI: http://dx.doi.org/10.1145/3022227.3022323

Keywords

Deep Learning; Machine Learning; Distributed Processing; Cloud Computing; Life-log Analysis

1. INTRODUCTION

The spread of various sensors and Cloud computing technologies have made it easy to acquire various life-logs and accumulate their data. As a result, many life-log analysis applications, which transfer data from cameras and sensors to the 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 analysis of sensor data in a Cloud and the efficient analysis processing in a Cloud. In these 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 limitation of network bandwidth between sensors and a Cloud. In addition, sending raw sensor data to a Cloud may introduce privacy issues.
Deep learning is a neural network technique widely used for analysis of images or videos. Deep learning makes it possible to automatically perform feature extraction from data; so, it has attracted attention for improving the accuracy and speed. There have been several deep learning frameworks such as Caffe [5], TensorFlow [1], and Chainer [10]. Caffe enables high-speed processing and provides trained network models. Preparing proper network definitions is one barrier to performing deep learning processing, but it is possible to use the network models provided by Caffe to easily perform experiments.
We propose distributed deep learning processing between sensors and a Cloud to reduce the amount of data sent to the Cloud and protect the privacy of users by sending preprocessed data. We also have developed this technique for the Caffe deep learning framework. We split a deep learning processing sequence of a neural network and performed dis-

tributed processing between the client side and the Cloud side in a pipeline manner. We compare the processing times of classification of three cases as follows: performing all processing on the client side, distributing the processing using the proposed method, and performing all processing on the Cloud side. In the experiments, we use the image data sets of CIFAR-10 [2] and ImageNet [8]. We also investigate the learning accuracy by varying a division point and its parameters to reduce the amount of data transferred to the Cloud. The experimental results show that the accuracy of deep learning with coarse-grain data is comparable to that with the default parameter settings, and the proposed distributed processing has performance advantages in the cases of insufficient network bandwidth using actual sensors and a Cloud environment.

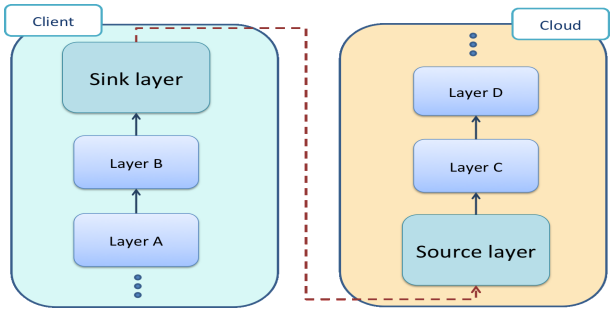

Figure 1: Proposed pipeline-based distribution method for deep learning.

2. DEEP LEARNING

Deep learning is a machine learning scheme using a neural network with a large number of middle layers. The neural network is an information system that imitates the structure of the human cerebral cortex. It is able to provide more exact recognition by extracting characteristics such as colors, shapes, and whole aspects, at the middle layers. Currently, it is widely used for the recognition of 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 determines the operation of the whole system through the communication betweenacross the modules. This approach can be expanded to new data formats and network layers. The core part of Caffe is written in C++, so, it is possible to use user-friendly image classification tools, such as the Jupyter Notebook implemented in Python, using the Caffe C++ API. In addition, Caffe is capable of high-speed processing because it corresponds to the GPU, and enabling the easy execution of experiments using the trained network models provided in the Caffe package.
Caffe constructs a network architecture called a convolutional neural network. The convolutional neural network, which is mainly applied to image recognition, usually repeats a convolution layer and a pooling layer in pairs to perform the basic calculation of image processing and then places full-connected layer. Convolution layers are used to compute dot products between the entries of the filter and the input image at any position and extract a characteristic gray structure represented by the filter from the image. Pooling layers take the square area of the input image and obtain a single pixel value 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 layer to be used in the Caffe network.

- Convolution
  The Convolution layer perform the convolution calculation. Here, it is necessary to set the number of filters and the kernel size representing the height and width of each filter. Optionally, it is possible to set the stride representing the interval to apply to the filters inof the input, the padding representing the number of pixels to be added to the edge of the input and the group representing the number of divisions in the channels.

- Pooling
  The Pooling layer performs the pooling calculation. It is necessary to set the kernel size and if possible, to set the methods of the pooling, the padding, and the stride. Examples of methods of pooling are max-pooling, which selects the maximum value from a set of pixel values, and mean-pooling, which calculates the average value.

- Local Response Normalization (LRN)
  The local response normalization layer performs a type of "lateral inhibition" by normalizing across local input regions.

- Inner Product
  The 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 as shown in Figure 1. We modified Caffe so that we can split a convolution neural network into two portion, the client side and the Cloud side. The client side and the Cloud side are the independent processes of Caffe. "Sink" is located at the end of the client side network, and terminates. "Source" is the starting point of the Cloud side network. Corresponding Sink and Source are connected by TCP/IP connection. Sink receives the data from upstream layer, transfers it to the paired Source and waits for the ACK from the Source. Source receives data from the Sink, sends ACK and then, forward data to downstream layers. Note that the client side prosess and the Cloud side process perform computation in parallel in a pipeline manner.
Examples of a configuration file of Sink and Source are shown in Figures 2 and 3. Sink specifies the host name and port number of the Cloud side process. Source specify the port number. Note that the Sink layer also defines the shape of the matrix. This is required since matrix size is necessary on the layer stack construction phase.
When one splits a network, more than one link could be cut. In such a case, one should set up a Sink-Source pair for each cut link. The pairs are identified by the port number. This approach makes it possible to protect the privacy of users by not sending raw data but rather sending feature

```
layer {
  name: "sink1"
  type: "Sink"
  bottom: "pool1"
  sink_param: {
          host_name: "server.example.com"
          port:      3000
    }
}
```

Figure 2: Example of configuration file of Sink.

```
layer {
  name: "source1"
  type: "Source"
  top: "pool1"
  source_param: {
            port:      3000
    }
    reshape_param{ shape { dim : [ 100, 32, 16, 16 ] } }
}
```

Figure 3: Example of configuration file of Source.

Table 1: Experimental environment

OS	Ubuntu 14.04LTS
CPU	Intel(R) Xeon(R) CPU W5590 @3.33 GHz (8 cores) × 2 sockets
Memory	8 Gbyte
GPGPU	NVIDIA GeForce GTX 980

values, and reducing the amount of transferred data between a sensor and a Cloud for low-bandwidth environments.

4. EXPERIMENTS

To indicate the effectiveness of the proposed method, we compare the processing times of the classification of three cases as follows: performing all processing on the client side, distributing the processing using the proposed method, and performing all processing on the Cloud side. In the experiments, we use the image data sets of CIFAR-10 [2] and ImageNet [8]. We also investigate the learning accuracy by varying a division point and its parameters to reduce the amount of data transferred to the Cloud. After we describe the network models and distribution methods for each of the datasets, we show the experimental results.

The experimental environment is as shown in Table 1. We use the same quality nodes in the experiments of investigating accuracies and those of comparison of processing times. We use one node in the former experiments and use two nodes on the client side and the Cloud side in the latter experiments. We use only a CPU on the client side and use a GPU on the Cloud side, and the network bandwidth between the two machines is 1 Gbps.

4.1 Datasets and the separation points of the distributed processing

4.1.1 CIFAR-10 Dataset

CIFAR-10 is a dataset in which images of 32 × 32 pixels are classified into 10 categories, and the network model of

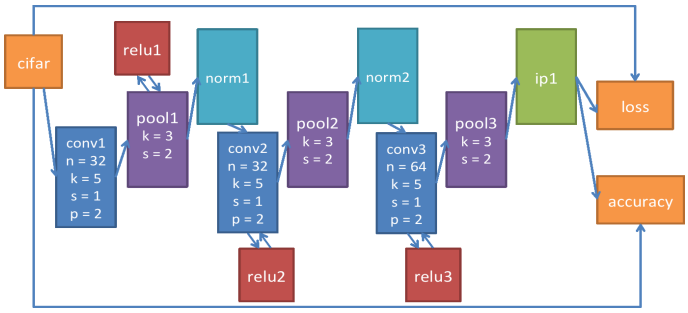

Figure 4: Network model for CIFAR-10.

Table 2: Parametars

n : num_output	number of filters
p : pad	width of padding
k : kernel_size	size of each filter
s : stride	interval to apply the filters
g : group	the number of division of the channels

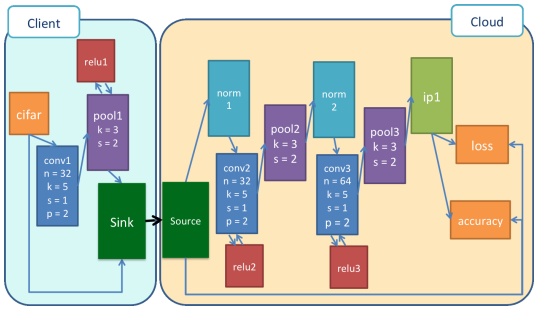

Figure 5: Distribution1 in the experiments using CIFAR-10.

one of the datasets is provided by Caffe. The structure of the network model is shown in Figure 4. The parameters defined in each layer are shown in Table 2.

Caffe stores and communicates data in 4-dimensional arrays as follows: the batch size, the number of channels and the two-dimensional image size. The channel parameters accord with the number of filters in the convolution layer just before that.

- Distribution1

  As shown in Figure 5, we split the network between the pool1 layer and the norm1 layer as distribution1. We reduce the number of the filters of the conv1 layer to reduce the amount of data. The numbers of filters of the conv2 layer and the conv3 layer are set to 32 and 64, respectively, which are the default values, varying the number of filters of the conv1 layer from 1 to 32. The correspondence between the number of filters and the amount of transferred data are shown in Table 3.

- Distribution2

  As shown in Figure 6, we split the network between the pool2 layer and the norm2 layer as distribution2. We reduce the number of the filters of the conv2 layer to reduce the amount of data. The numbers of filters

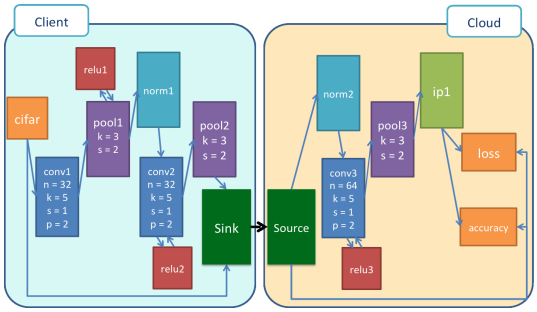

Figure 6: Distribution2 in the experiments using CIFAR-10.

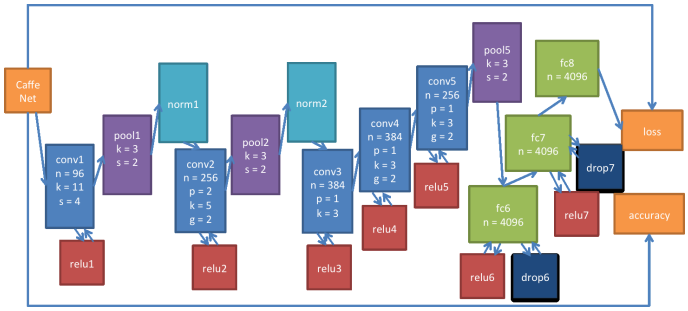

Figure 7: Network model for ImageNet

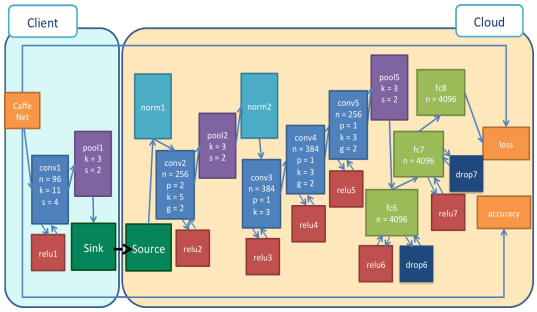

Figure 8: Distribution1 in the experiments using ImageNet.

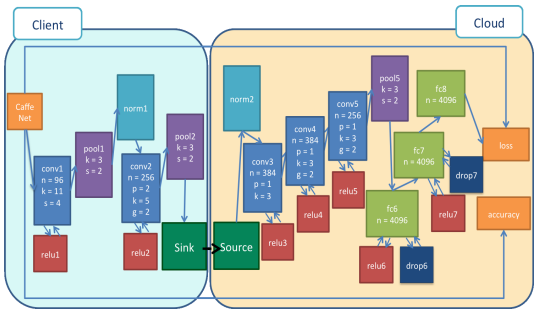

Figure 9: Distribution2 in the experiments using ImageNet.

of the conv1 layer and the conv3 layer are set to 32 and 64, respectively, which are the default values, varying the number of filters of the conv2 layer from 1 to 32. The correspondence between the number of filters and the amount of transferred data are shown in Table 3.

4.1.2 ILSVRC2012 Dataset

ILSVRC (ImageNet Large Scale Visual Recognition Challenge) is a competition that evaluates algorithms for object detection and image classification at a large scale. The test data set 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 of the is provided as CIFAR-10, as shown in Figure 7.

The batch size for the test is set to 256, so when the parametar is set to the default value, the amount of data is (256 $\times$ 3 $\times$ 227 $\times$ 227) bytes at the beginning, and it becomes (256 $\times$ 96 $\times$ 55 $\times$ 55) bytes after the conv1 layer, where the number of filters is 96 and the stride is 4. Then, the amount of data after the pool1 layer, where the stride is 2, becomes (256 $\times$ 96 $\times$ 27 $\times$ 27) bytes, which is about one-half of the raw data. Identically, the amount of data becomes approximately a one-third after the pool2 layer. Here, we show two distribution methods utilized in the experiments in this paper, considering the amount of data during communication.

- Distribution1
  We split the network between the pool1 layer and the norm1 layer as distribution1 (Figure 8). The amount of data during the communication is approximately half of raw data, and we consider to further reducing the amount of data by changing the number of filters at the conv1 layer. The numbers of filters are set to 72 and 48 in this paper, which are three-fourths and one-half of default value, respectively.

- Distribution2
  We split the network between the pool2 layer and the norm2 layer as distribution2 (Figure 9). The amount of data during the communication is approximately one-third of the raw data, and we consider further reducing the amount of data by changing the number of filters in the conv2 layer. The numbers of filters are set to 192 and 128 in this paper, which are three-fourths and one-half of the default value, respectively.

4.2 Learning acuracies upon varying the number of filters

The reduction of the amount of transferred data between the layers of the neural network may decrease the accuracy of recognition; so, we investigate the accuracies when we change the number of filters and thereby reduce the amount of transferred data.

In the experimets using CIFAR-10, the correspondence between the number of filters and the amount of transferred data is shown in Table 3. The experimental results of distribution1 using CIFAR-10 is shown in Figure 10, and that of distribution2 is shown in Figure 11. The horizontal axises represent the filter numbers, and the vertical axises represent the accuracy of the identification. We can see that the accuracies converge when the numbers of filters are small, as shown in Figure 10 and Figure 11. Even in the case of the number of filters being 4 at the conv1 layer, the accuracy is maintained at 73.35%, which is comparable to the result of the default state, 78.11%, while the amount of transferred data in the case of 4 is one-third of the raw data. In the

Table 3: Correspondence between the number of filters in the conv1 layer and the conv2 layer and the amount of data for communication 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

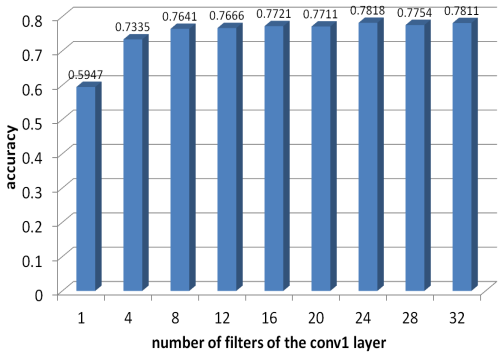

Figure 10: Accuracy upon varying the number of filters of the conv1 layer using CIFAR-10.

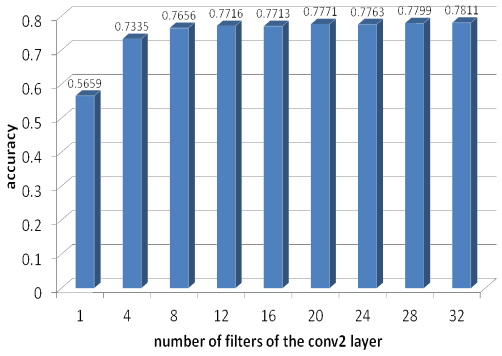

Figure 11: Accuracy upon varying the number of filters of the conv2 layer using CIFAR-10.

case of the number of filters being 4 at the conv2 layer, the accuracy is maintained at 73.35%, while the amount of transferred data is one-twelfth of the raw data.
In the experiments using ImageNet, the accuracy is maintained at 56.8% in the case of 72 at conv1 layer and 56.3% in the case of 48, compared to 57.4% in the case of the default value. And the accuracy is maintained at 57.0% in the case of 192 at conv2 layer and 56.4% in the case of 128.
Hence, we can see that high accuracy can maintained even if we reduce the amount of transferred data by reducing the number of filters of the convolution layers.

4.3 Comparison of Processing Times

We show the effectiveness of the proposed method by measuring the processing times of the identification of 1 batch using two machines as the client side and the Cloud side, respectively. We compare three cases as follows: (1) performing all processing on the client side, (2) distributing processing between the client and Cloud sides using the proposed

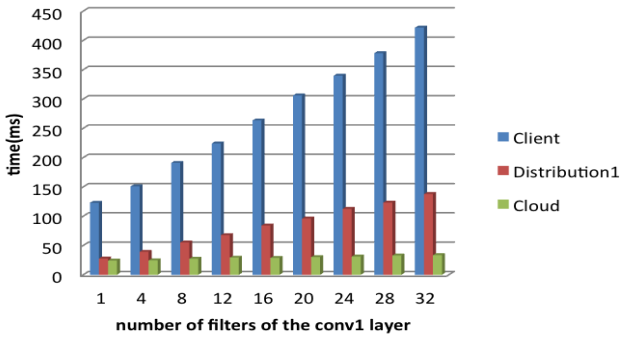

Figure 12: Processing times varying the number of filters at the conv1 layer using CIFAR-10 (1 Gbps).

method, (3) performing all processing on the cloud side. No data are transferred between the client and the Cloud in case (1), while processed and filtered data are sent to the Cloud in case (2), and the raw image data are sent to the Cloud in case (3). In cases (2) and (3), the client side and the Cloud side synchronously work, so we use the processing times measured in the client side, including connection times and waiting times. In all of the experiments, we use only a CPU on the client side and use a GPU on the Cloud side. We use the same quality nodes on the client side and the Cloud side as shown in Table 1, and the network bandwidth between the two machines is 1 Gbps. We use PSPacer [9] for network bandwidth control in order to represent various sensor and Cloud netowork environments.

4.3.1 Experiments using CIFAR-10

We set the network bandwidth between the two machines to 1 Gbps and 10 Mbps and measure the processing times by varying the number of filters of the conv1 layer.

The results of Distribution1 are shown in Figure 12 and 13 and those of Distribution2 are shown in Figure 14 and 15. Client (CPU), Distribution and Cloud (GPU) refer the results in cases (1), (2) and (3), respectively. In figure 12, because of the reduced processing on the client side, the results of case (2) are faster in comparison with those of case (1). At this time, the case (3) are superior to the other cases because deep learning processing for images takes more than the time of transmission between the client and the Cloud. However, in the 10 Mbps environment, considering the communication environment between general homes and a Cloud, the results of case (2) are faster than those of case (3) when the number of filters is set to less than 12, so that the amount of transferred data is smaller than the raw data. Case (1) is not realistic because of the limitation of the resources on the sensor side, so distributed processing is effective in an actual environment.

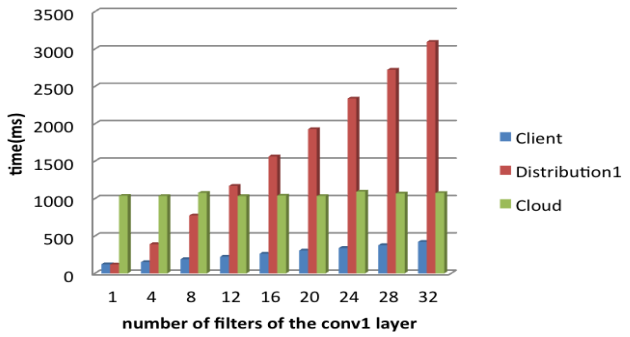

Figure 13: Processing times varying the number of filters at the conv1 layer using CIFAR-10 (10 Mbps).

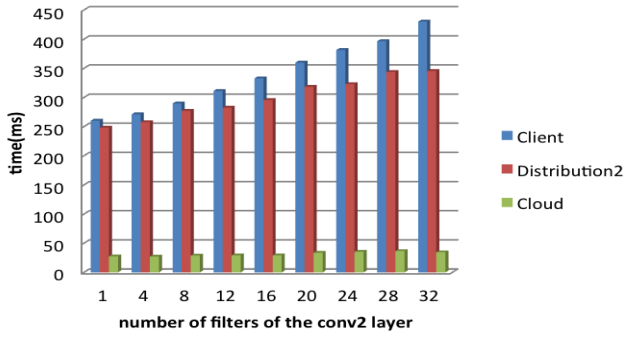

Figure 14: Processing times varying the number of filters at the conv2 layer using CIFAR-10 (1 Gbps).

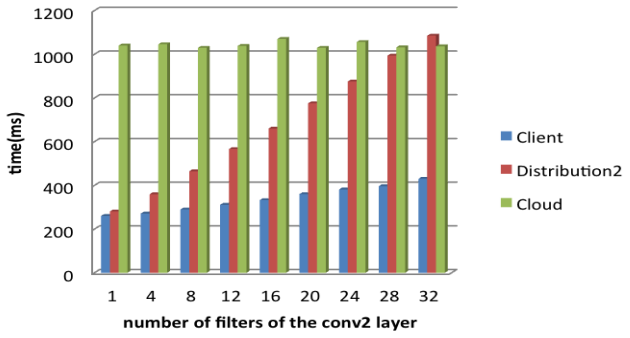

Figure 15: Processing times varying the number of filters at the conv2 layer using CIFAR-10 (10 Mbps).

Figure 14 shows that in distribution2, the results of case (2) take longer than in distribution1 because the deep learning processing for images of the client is more extensive than distribution1, and the processing time is longer upon increasing the amount of data from the conv1 layer to the conv2 layer for setting the number of filters of the conv1 layer to a default value. Case (3) is superior to the other cases as with distribution1, but in the 10 Mbps environment, the results of case (2) are faster than those of case (3) because the amount of transferred data are smaller in

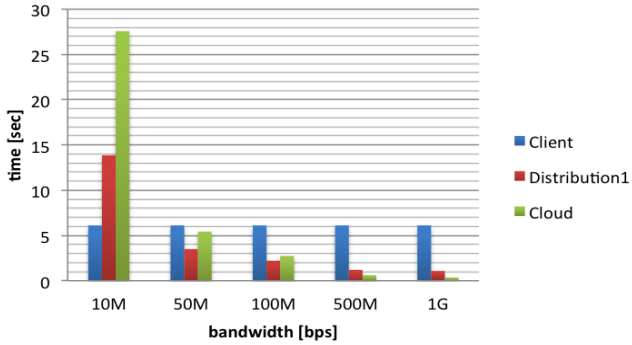

Figure 16: Processing time of distribution1 for setting the number of filters of the conv1 layer to the default value using ImageNet.

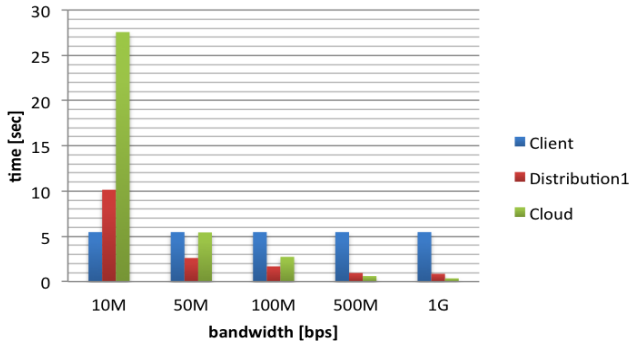

Figure 17: Processing time of distribution1 for setting the number of filters of the conv1 layer to 72 using ImageNet.

distribution2. As with the results of distribution1, case (1) is faster than the other cases but not realistic, so distributed processing is effective.

4.3.2 ImageNet

We measure the processing times by fixing the number of filters and changing the network bandwidth between the two machines from 1 Gbps to 10 Mbps.

Figures 16, 17 and 18 show the results of distibution1, setting the number of filters of the conv1 layer to 96, 72 and 48, respectively and Figures 19, 20 and 21 show the results of distibution2, setting the number of filters of the conv2 layer to 256, 192 and 128, respectively.

In both experiments, the results of case (3) are faster than those of the other cases when the network bandwidth between two machines is 1 Gbps. However, it is confirmed that for 10 Mbps and 50 Mbps, the results of case (3) take longer than those of case (2) because the raw data are large, and it takes a longer time for communication, so the total processing time is longer. In addition, the processing time in the distributed processing can be further reduced by reducing the number of filters. When the network bandwidth between the two machines is set to 10 Mbps, the result of

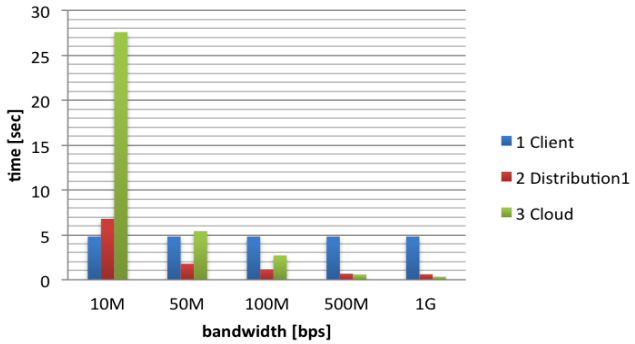

Figure 18: Processing time of distribution1 for setting the number of filters of the conv1 layer to 48 using ImageNet.

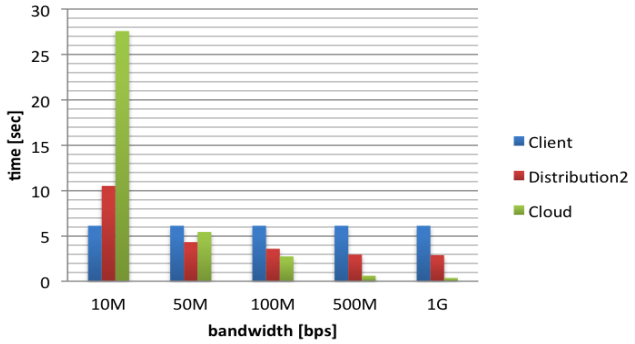

Figure 19: Processing time of distribution2 for setting the number of filters of the conv2 layer to the default value using ImageNet.

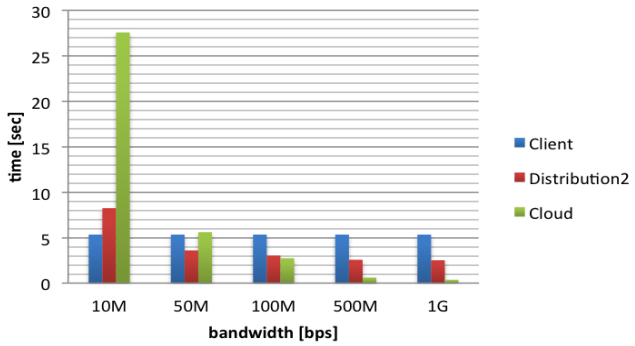

Figure 20: Processing time of distribution2 for setting the number of filters of the conv2 layer to 192 using ImageNet.

case (2) of distribution2 is faster than those of the other cases, and when the network bandwidth between the two machines is set to 50 Mbps or 100 Mbps, the result of case

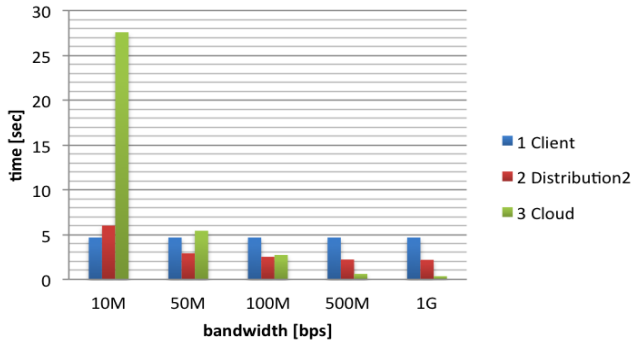

Figure 21: Processing time of distribution2 for setting the number of filters of the conv2 layer to 128 using ImageNet.

(2) of distribution1 is faster. So, we can see that the efficient distribution method is dependent on the network bandwidth and the performance of the machines on the client side and the Cloud side.

5. RELATED WORK

Neural networks have been widely used to precisely identify and classify patterns [6] [7]. However, many of the frameworks perform at high speed by utilizing the GPU in a single computer.

In [4], Caffe is executed in various environments in Clouds. In this research, a fully compatible end-to-end version of the popular framework Caffe with rebuilt internals is developed. They achieved a 6.3 × throughput improvement over Caffe on 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 enables us to efficiently train hybrid CPU-GPU systems for CNNs. However, this enables the performance of high-speed processing only in Clouds, but not executed in a distributed environment. In [11], automatic font identification using a neural network in the client-server environment is developed. However, this research presents a system running on a thin-client. Only I/O processing is performed on the client side, and recognition using the neural network is performed on the server side. Compared to this research, our research work is different because our system is distributed between the client side and the Cloud side. We mentioned the DIANNE middleware framework as related work [3]. While a normal neural network consists of an input layer, 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 neural network components on a large number of heterogeneous devices. On the other hand, the amount of transferred data and the processing time for the distributed method are not considered in this work. Therefore, we aim to reduce the processing time by considering the amount of transferred data and propose an efficient framework for the process of the life-log analysis for general consumers.

6. CONCLUSIONS

We propose a pipeline-based distributed processing for deep learning and implement the distributed processing of the deep learning framework Caffe for the purpose of the sensor data analysis process, considering privacy and the network bandwidth. When we take into account a realistic network bandwidth between general homes and a Cloud, it takes time to transfer raw sensor data, so the effectiveness of the proposed method is proven. We observed that it is possible to maintain a high accuracy and perform efficient processing even if we reduce the amount of transferred data from a sensor to a Cloud by reducing the number of filters of the convolution layers.
In the future, we will perform sexperiments using machines of realistic performance as a client for general homes and using video data.

7. ACKNOWLEDGMENT

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

8. REFERENCES

[1] 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, 2015. http://download.tensorflow.org/paper/whitepaper2015.pdf. pp. 1-19.

[2] K. Alex, V. Nair, and G. Hinton. The cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html(accessed December 27, 2015).

[3] E. De Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, S. Leroux, and P. Simoens. Dianne: Distributed artificial neural networks for the internet of things. In *In Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT*, M4IoT 2015, pages 19–24, New York, NY, USA, 2015. ACM.

[4] S. Hadjis, F. Abuzaid, C. Zhang, and C. Ré. Caffe con troll: Shallow ideas to speed up deep learning. In *Proceedings of the Fourth Workshop on Data Analytics in the Cloud*, DanaC'15, pages 2:1–2:4, 2015.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM'14)*, pages 675–678, 2014.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[7] S. Pierre, E. David, Z. Xiang, M. Michaël, F. Rob, and L. Yann. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *In Proceedings of International Conference on Learning Representations*, 2013. 15 pages.

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

[9] R. Takano, T. Kudoh, Y. Kodama, and F. Okazaki. High-resolution timer-based packet pacing mechanism on linux operating system. *IEICE Transactions on Communications*, E94.B(8):2199–2207, 2011.

[10] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015. 6 pages.

[11] Z. Wang, J. Yang, H. Jin, J. Brandt, A. Agarwala, Z. Wang, Y. Song, J. Hsieh, E. Shechtman, S. Kong, and T. S. Huang. Deepfont: A system for font recognition and similarity. In *In Proceedings of the 23rd ACM international conference on Multimedia*