International Journal of Advanced Smart Convergence Vol.10 No.4 278-288 (2021)
http://dx.doi.org/10.7236/IJASC.2021.10.4.278

IJASC 21-4-33

Implementation of AIoT Edge Cluster System via Distributed Deep Learning Pipeline

Sung-Ho Jeon*, Cheol-Gyu Lee*, Jae-Deok Lee*, Bo-Seok Kim*, Joo-Man Kim†

**Undergraduate Students, Dept. of Applied IT and Engineering, Pusan National University, Pusan*
†Professor, Dept. of Applied IT and Engineering, Pusan National University, Pusan
e-mail : {tjdgh0715,cjfrb0811,nellcome,pos05073}joomkim@pusan.ac.kr

Abstract

Recently, IoT systems are cloud-based, so that continuous and large amounts of data collected from sensor nodes are processed in the data server through the cloud. However, in the centralized configuration of large-scale cloud computing, computational processing must be performed at a physical location where data collection and processing take place, and the need for edge computers to reduce the network load of the cloud system is gradually expanding. In this paper, a cluster system consisting of 6 inexpensive Raspberry Pi boards was constructed to perform fast data processing. And we propose "Kubernetes cluster system(KCS)" for processing large data collection and analysis by model distribution and data pipeline method. To compare the performance of this study, an ensemble model of deep learning was built, and the accuracy, processing performance, and processing time through the proposed KCS system and model distribution were compared and analyzed. As a result, the ensemble model was excellent in accuracy, but the KCS implemented as a data pipeline proved to be superior in processing speed..

Key words: *Distributed Data pipeline, Deep Learning, IoT, Edge Computing, Kubernetes, Docker*

1. Introduction

Today, deep learning is being applied and used in various fields such as image recognition, natural language processing, and anomaly detection. In addition, as the size of data to be processed and the size of models that perform calculations increased rapidly with the development of data processing technology, advanced artificial intelligence technologies such as GPT-3 appeared.[1] For deep learning operations, it is important to have a computing system environment that can efficiently perform operations not only in software but also in hardware environments with limited performance. Distributed networks have been proposed as a solution to this, and studies are underway to utilize these technologies in computing systems with efficiency and performance and in AI fields. [2,3] Distributed deep learning shows superior performance compared to single processing. was proven, and research was conducted to operate in a multi-processing environment by constructing a sophisticated model. [4.5] However, one of the disadvantages of the distributed network system

Manuscript Received: December. 6, 2021 / Revised: December. 8, 2021 / Accepted: December. 9, 2021
†Corresponding Author: joomkim@pusan.ac.kr
Tel:*** - **** - **** Fax: +82-55-350-5415
Professor, Dept. of Applied IT and Engineering, Pusan National University, Pusan, Korea

for implementing this is that the construction cost of the system itself is very high. Performance or efficiency cannot be expected when a distributed computing environment is built using inexpensive IoT devices, not GPUs or TPUs, which are mainly used in distributed network system environments. Accordingly, research on how to build a distributed network using relatively inexpensive hardware is currently underway, and the need for additional research on how to build a low-cost distributed learning system apart from a large-scale model in the future is also emerging. [6]

IoT is a key technology of the future, and the size of the IoT market is growing every year.[7] Recently, in order to advance the IoT system, many studies are being conducted to build an autonomous AIoT system by combining AI with IoT.[8] The key to combining IoT and AI is to process a large amount of AI computation within the user's desired reaction time. However, most IoT devices do not have the computing power to satisfy the user's desired reaction time by performing AI calculations. Although cloud services have been commercialized to solve this computing power problem, cloud services have limitations in long communication delays due to high RTT and low network bandwidth.[9] To overcome the limitations of these services, edge computing technology was introduced where data collection takes place before accessing the cloud.[10] Edge computing is considered as one of the important future technologies for IoT system computing because it performs calculations directly on the device itself rather than on the cloud method that performs centralized calculation processing. [11,12,13] However, since IoT devices use various operating systems, the need to separately build a processing system for performing edge computing in various devices with heterogeneity is also raised. Edge computers require processing performance for continuous data collection, processing, and storage, and processing and analysis of big data collected and stored from external sensor devices [14] Analytical techniques and processing performance are required. Proven processing performance in a distributed processing environment by connecting multiple Raspberry Pi.[15]

In this paper, we built an edge computing system platform (KCS) that mounts virtual containers on IoT devices through Docker for 6 Raspberry Pi boards and clusters virtualized nodes through Kubernetes. As a deep learning model to be mounted on the cluster, a transfer learning-based model was built using MobilenetV2 specialized for embedded devices, and then mounted on the pipeline connecting the nodes in the cluster.

2. Related Studies

2.1 Docker

Docker is an open source virtualization platform for building applications based on containers.[16] Docker installed in the operating system kernel has a structure similar to that of a virtual machine (VM), but there is no physical emulation. Due to these characteristics, it is possible to create various containers on one host, and by operating it, it has the advantage of building a virtual environment that is more efficient than a virtual machine.

2.2 Kubernetes

Kubernetes is a management system that provides automatic deployment and scaling of container-level applications.[17] It aims to provide a platform for automating the deployment, scaling and operation of application containers across hosts in multiple clusters. It has the nature of working with a set of container tools, including Docker. It has the advantage of easy management and search by grouping the containers that

make up the application into logical units. Kubernetes gradually rolls out changes to the application or its configuration while monitoring the health of the application to ensure that not all instances are killed at the same time, and rolls back the changes when issues arise. Containers in Kubenatis are called PODs, and you can give these PODs a unique IP address and a single DNS for a set of PODs and perform load balancing. It also restarts failed containers, replaces and reschedules containers when a node dies. It also shuts down containers that do not respond to custom health checks, not notifying clients until they are ready to serve.

2.3 CNN

CNN, also called convolutional neural network, is a neural network specialized in processing data such as images arranged in a fixed grid form.[18] It has greatly contributed to image search services, autonomous vehicles, and automatic image classification systems, and is used in fields such as speech recognition and natural language processing in addition to visual fields. In CNN, the convolutional layer automatically finds the most useful filter (convolutional kernel) through learning, and the upper layer connects them to learn more complex patterns.

2.4 MobilenetV2

MobilenetV2, announced by Google in 2019, is a model that performed better than MobilenetV1, which effectively reduced the amount of computation and the number of model parameters by introducing the concept of Depthwise Separable Convolution. Due to the nature of the lightweight model, it is suitable for use in limited environments such as mobile devices, and based on these characteristics, it is actively used in the embedded environment. MobilenetV2 reduces the dimension when constructing a bottleneck structure that maps data to a lower dimension through the Linear Bottleneck Layer, but enables the mainfold [19].

Figure 1 shows the core operating mechanism of the MobilenetV2 model, the deep learning model used in this paper. In MobilenetV2, Inverted Residual Block works in the opposite mechanism to Resnet's Residual Block, which is a representative CNN high-performance model, and through this, the compressed narrow layer is used as a skip connection, thereby reducing memory usage [20]. However, MobilenetV1 and MobilenetV2 have the disadvantage that there is a trade-off relationship between model size and performance [21]. Therefore, this paper is designed to preserve the size of the MobilenetV2 model as much as possible and guarantee improved performance using transfer learning.

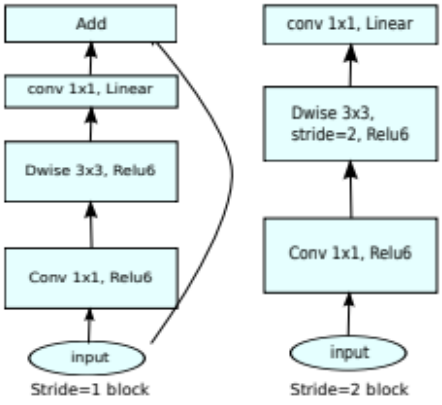


Figure 1. The core operating mechanism of MobilenetV2[19]

2.5 Distributed Deep Learning

The computational resources of a single computer are not sufficient to run large-scale neural networks. A learning method that has emerged to solve this problem is distributed learning. There are two types of distributions in deep learning. A parallel distribution method of data in which each input sample is processed by a different computer, and a model distribution method in which one data point is processed by multiple computers. In this case, the deep learning model is divided into layers and deployed to multiple computers, so the computers run different parts of the model. [22,23]

3. System Design

3.1 Edge cluster for KCS

In this paper, using 6 Raspberry Pi 4Bs, it consists of 1 master node and 5 slave nodes. By mounting a virtual container on each node through Docker, the virtualized node is configured to cluster multiple Dockers by Kubernets. Doker, which provides the same environment as a method to cope with the heterogeneity of various devices in the Internet of Things environment, and the application of Kubernetes to manage these same virtualized nodes are the core structures of this paper. For each container in the cluster, an environment for processing AI jobs was built using python3.7 and tensorflow2.4.

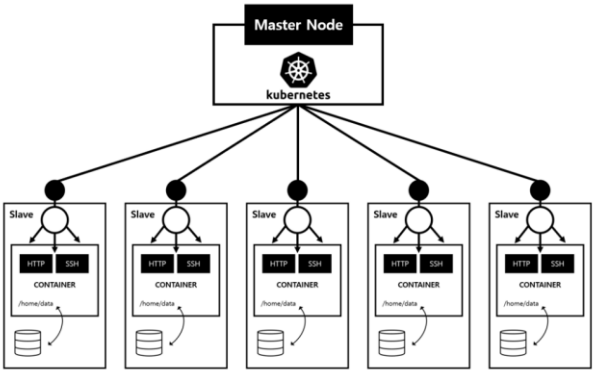


Figure 2. The structure AIoT Edge Kubernetes cluster

In the study conducted in this paper, Jupyter Notebook and SSH service are load-balanced in containers. By servicing Jupyter Notebook, it facilitates maintenance of the AI model and execution environment through external port access, and provides SSH service for file transfer and command transfer between nodes. At this time, multiple ports are used in one container, and a load balancer is applied to connect the ports according to the purpose of the service. Mount the /home/data directory in the container to the /mnt/data directory of the Raspberry Pi to make it easy to use the file system inside and outside the container. Figure 2 shows the overall structure of KCS proposed in this paper.

3.2 MobilenetV2 distributed pipeline model based on transfer learning

The Raspberry Pi device lacks computational power to use the image processing model. Therefore, in this paper, we propose a distributed deep learning pipeline model as a way to ensure fast processing time by

installing it on a Raspberry Pi cluster. Among deep learning models, we learn MobilenetV2, a CNN model specialized for embedded environments. At this time, in order to supplement the trade-off relationship between the performance of the MobilenetV2 model and the amount of computation, learning is carried out using the transfer learning technique. By applying the model distribution technique to the trained model, the layer is divided, and data pipelining is performed by loading it on each node of the cluster.

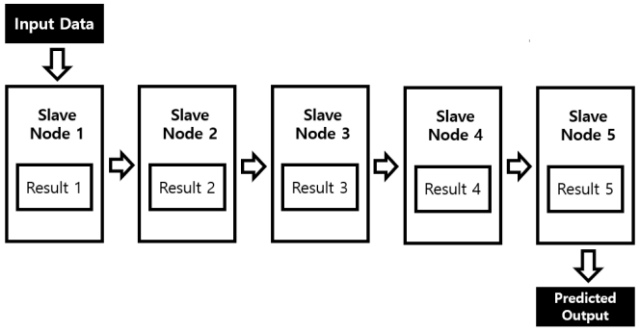


Figure 3. Structure of the Parallel Deep Learning Pipeline Model

Figure 3 shows the structure of the distributed deep learning pipeline model proposed in this paper. Nodes in the cluster constituting the pipeline can communicate and transmit data between nodes using SSH communication. Analyze the distribution of the number of parameters and the amount of computation for each layer of MobilenetV2. After that, considering the amount of computation, the entire layer of the model is divided into 5, and then allocated to each slave node in the cluster. When input data enters the pipeline, it goes through the layers of the model in turn and performs calculations. Then, the execution result is transmitted to the input of the next node and the next operation is performed simultaneously. The next node performs an operation based on the received weight and performs the same operation as the previous node. The last node constituting the pipeline outputs the final execution result of the operation as an output.

4. System Implementation and Performance Evaluation

4.1 Implementing a Kubernetes Edge Cluster

An edge cluster consisting of 1 master node and 5 slave nodes was built using 6 Raspberry Pi 4Bs. Kubernetes was used as the cluster system.

```
pi@commendcenter:~ $ kubectl get nodes
NAME            STATUS   ROLES                  AGE   VERSION
commendcenter   Ready    control-plane,master   77d   v1.22.1
scv1            Ready    <none>                 77d   v1.22.1
scv2            Ready    <none>                 49d   v1.22.2
scv3            Ready    <none>                 49d   v1.22.2
scv4            Ready    <none>                 49d   v1.22.2
scv5            Ready    <none>                 49d   v1.22.2
```

Figure 4. Kubernetes node connection status

Figure 4 shows the results of checking the connection status of each node through the command after building a cluster using Kubernetes. First, build an image that will be used to build an environment for

performing operations on each node in the cluster using Docker. At this time, the image was built by creating a new image called cjfrb0811/pi-tensorflow:2.4.0 and pushed to DockerHub. After that, a POD container was created and distributed by pulling the cjfrb0811/pi-tensorflow:2.4.0 image from Kubernetes. 'RestartPolicy = Always' is set so that re-execution is possible when a failure occurs in the container during the POD deployment process.

tensorflow-deployment1~5 are fixedly assigned to each node and provide an environment for implementing data pipelining. In the cluster constructed in this paper, the real node and the POD, which is a virtual environment, are mounted to each other to connect the storage space. Public Volume was allocated 5GB of storage space for Node, and it was set to be allocated up to 3GB when requesting a claim. Through this, the real space "/mnt/data" of the node and the virtual space "/home/data/" of the POD can share the storage space.

```
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
tensor-deployment    1/1     1            1           43h
tensor-deployment1   1/1     1            1           43h
tensor-deployment2   1/1     1            1           43h
tensor-deployment3   1/1     1            1           43h
tensor-deployment4   1/1     1            1           43h
tensor-deployment5   1/1     1            1           43h
```

Figure 5. Deployment of the Kubernetes POD

Figure 5 shows the result of deploying the POD to each node in Kubernetes and checking the deployed status. When accessing the POD from the outside, load balancing was performed to allow access through the HTTP method and the SSH method. As a method for this, an L4 load balancer that distributes the port load was applied and used. The HTTP method provides services for simple file management, POD-specific deep learning model execution and testing through Jupyter Notebook. The SSH method provides services for file transmission and reception, remote execution of files inside the POD, and return of execution results. Both ports use the TCP protocol, and all external access IPs are designated as the IP of the general node. Figure 6 shows the result of checking the port and connection status of each node after deploying the service in Kubernetes.

```
kubernetes    ClusterIP      10.96.0.1        <none>           443/TCP
tensor-svc    LoadBalancer   10.109.192.237   164.125.234.90   8888:32315/TCP,5000:30496/TCP
tensor-svc1   LoadBalancer   10.104.101.204   164.125.234.84   8888:30792/TCP,2222:30237/TCP
tensor-svc2   LoadBalancer   10.108.250.221   164.125.234.85   8888:30444/TCP,2222:31662/TCP
tensor-svc3   LoadBalancer   10.107.134.63    164.125.234.86   8888:30306/TCP,2222:30016/TCP
tensor-svc4   LoadBalancer   10.105.76.242    164.125.234.87   8888:30446/TCP,2222:31504/TCP
tensor-svc5   LoadBalancer   10.100.213.37    164.125.234.88   8888:31064/TCP,2222:31004/TCP
```

Figure 6. Kubernetes Service

4.2 MobilenetV2 learning based on transfer learning

In this paper, Kaggle's PlantVillage dataset was used as training data for the MobilenetV2 model. The dataset consists of a total of 16023 tomato pest image datasets, divided into 10 normal/pest classes. The dataset has severe data imbalance, which can reduce the reliability of training and cause overfitting in model training. Figure 7 is a chart showing the distribution by class of the dataset before preprocessing.

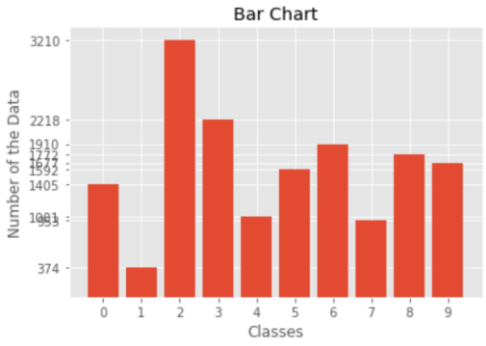
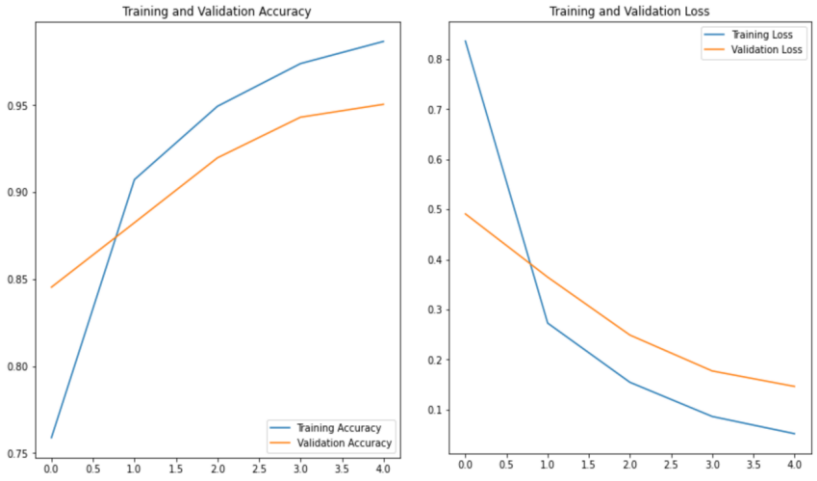


Figure 7. Distribution of pre-processing data by class

Before performing model learning, a class imbalance resolution task through data augmentation was performed as a data preprocessing task. As a preprocessing technique, RandomRotation, RandomFlip, RandomCrop, and RandomZoom functions, which are data preprocessing functions of the Keras module, were used. As a result of performing data pre-processing through this, a total of 100,000 image datasets consisting of 10,000 sheets per class was constructed. Figure 8 is a chart showing the distribution by class of the dataset after preprocessing.

Figure 8. Distribution of data after pre-processing by class

When learning MobilenetV2, after setting the batch size to 32, model preprocessing was performed through the preprocess_input() method. To optimize the data input step, caching and prefetching using AUTOTUNE were performed. The training data was mixed with the train dataset with a buffer of size 100. GlobalAveragePooling was used as the pooling technique, and dropout was applied. As the activation function for the output, the softmax function was used in consideration of the fact that the data distribution according to the class has a multi-sister distribution. In the learning process, the pooling layer, dropout layer, and output layer were first learned by freezing the original model base model (mobilenetV2). At this time, Adam was used as the optimizer, and CategorialCrossentropy was used as the loss function. In the initial training, the epoch was set to 5. As a result of training, the valid loss was 0.2864 and the valid accuracy was 0.9053. After that, during transfer learning, the base model was unfreeze and then training was carried out. In this case, Adam was used as the optimizer, and the learning rate was set to 0.00001. As the loss function, CategorialCrossentropy was used.



(a) Valid Accuracy (b) Valid Loss

Figure 9. Graph of Training Results

Figure 9(a) is a graph showing the training degree and validation accuracy of a model that has performed transfer learning. As a result of training, the valid loss was 0.1465 and the valid accuracy was 0.9505. When the changes in accuracy and loss values were checked in graph form using the matplotlib module, the train value and valid value graph came out with a similar distribution, and based on this, it was confirmed that overfitting did not occur. In addition, as a result of calculating the F1 score, 0.93 was obtained, confirming that the model showed excellent performance. Figure 9(b) is a graph showing the training and validation errors of the trained model.

4.3 Implementation of distributed deep learning pipeline by model partitioning

After performing transfer learning, a distributed deep learning pipeline was implemented by performing model segmentation. After analyzing the entire layer of MobilenetV2, a total of 157 layers of the entire model were divided into 5 models, each 27/27/36/53/14 considering the amount of computation. The divided model was assigned to 5 slave nodes in the cluster. We implemented a distributed deep learning pipeline model that returns a weight value through Gradient Descent and Batch Normalization for each unit of layer where the operation is executed, designates it as the output of the node, and transmits this weight value as the input of the next node. Figure 10 shows the actual construction of KCS equipped with a distributed deep learning pipeline.

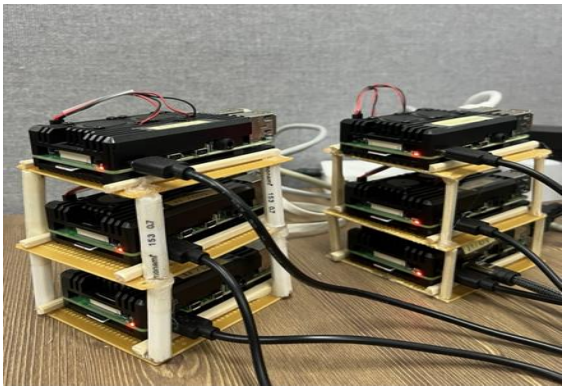


Figure 10. The actual Raspberry Pi cluster

4.4 Comparative evaluation for model performance

In this paper, an ensemble model was additionally implemented to compare the performance of the distributed deep learning pipeline model installed in KCS.

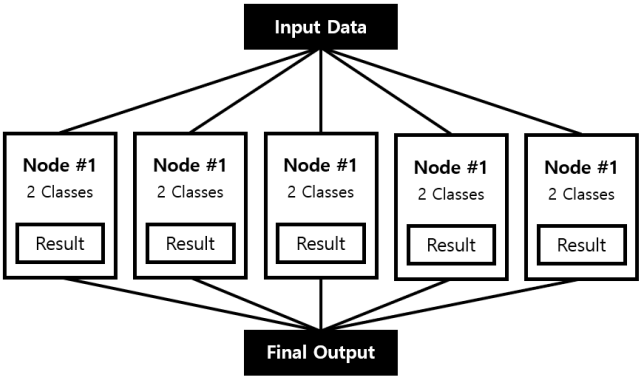


Figure 11. Structure of the Ensemble processing model

Figure 11 shows the structure of the ensemble model built for performance verification in this paper. In the full deep learning model that classifies 10 classes in one device, one input data is classified into two classes at each node, and then the result is changed to an ensemble model that sums the results. In addition, the MobilenetV2 model also reduced I/O classes to two to perform binary classification. In this case, the activation function used the sigmoid function considering that the output data according to the class follows the Bernoulli distribution. When input data enters the cluster, the same input data is transmitted to the model of each node through SSH communication, and then the classification operation is performed. After collecting the results of each operation, the final prediction of the model with the highest performance is returned. At this time, in order to prevent the possibility of misclassification, the model's performance is verified by comparing the class predicted by the model with the class of the input data once more.

Table 1. Comparision of performance based on type of construction models

comparative model	val_accuracy	F1-Score	Processing Time
KCS	95%	0.93	60s
ensemble	97%	0.94	260s
single processor	92%	0.91	580s

Table 1 shows the results of comparing the performance of the KCS and control ensemble model proposed in this paper and the model in a single processor environment. As a result of the experiment, it was confirmed that in the edge computing environment based on ensemble, the performance was slightly higher than that of KCS, but the processing time was more than 5 minutes. On the other hand, it was confirmed that the KCS model has slightly lower performance than the ensemble model, but has a processing time that is about 70% faster. When compared with the single processor environment, it was confirmed that the KCS model has 90% faster processing time. Through this, it was confirmed that the distributed deep learning pipeline model can guarantee good performance based on the fastest processing time in order to mount AI within a fast processing time that can satisfy users in an edge computing environment.

5. Conclusion

The occurrence of a lot of data in the IoT environment saturates the cloud environment, and thus the service that does not depend on time and place could not be satisfied. In this paper, we proposed a high-performance edge computer by reducing the cloud load, clustering the Raspberry Pi for high-performance processing at the data generation point, and using a distributed deep learning technique. In this paper, we propose and implement a Kubernetes-based edge cluster computing environment as a way to perform deep learning operations within the processing time that users can satisfy in the AIoT environment. In addition, a distributed deep learning pipeline model optimized for multi-processing was proposed and implemented through the model distribution of deep learning with a large amount of computation. In addition, it was demonstrated that an AIoT platform that guarantees low latency and high processing speed when performing deep learning operations in the cluster system (KCS) implemented in this paper can be implemented. If the KCS proposed in this paper is mounted on a high-performance GPU processor-based cluster, it is expected to show very good performance even when a model with a large amount of computation is loaded and executed.

Acknowldegment

This work was supported by a 2-Year Research Grant of Pusan National University.'

References

[1] Tom B.Brown et al. "Language Models are Few-Shot Learners", arXiv preprint arXiv:2005.14165, (2020).
[2] Park, Yoomi et al. "Deep Learning Model Parallelism", ETRI.2018.J.330401 (2018),
DOI: 10.22648/ETRI.2018.J.330401
[3] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, Phil Gibbons "PipeDream: Fast and Efficient Pipeline Parallel DNN Training", arXiv:1806.03377, (2020), https://arxiv.org/pdf/1806.03377.pdf

[4] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng, "Large Scale Distributed Deep Networks", Advances in Neural Information Processing Systems 25 (NIPS 2012)
[5] TAL BEN-NUN and TORSTEN HOEFLER, ETH Zurich, "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis" arXiv:1802.09941, (2018), https://arxiv.org/pdf/1802.09941.pdf
[6] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, Ng Andrew, "Deep learning with COTS HPC systems", Proceedings of the 30th International Conference on Machine Learning, PMLR 28(3):1337-1345, (2013), https://proceedings.mlr.press/v28/coates13.html
[7] Falguni Jindal, Savy Mudgal, Varad Choudhari, Prathamesh P. Churi, "Emerging trends in Internet of Things", 2018 Fifth HCT Information Technology Trends (ITT), (2019)
[8] Jin-Young Kim, Isaac Sim, Sung-Hoon Yoon, "Artificial Intelligence-based Classification Scheme to improve Time Series Data Accuracy of IoT Sensors", The Journal of the Institute of Internet, Broadcasting and Communication ::, Vol.21 No.4 | pp.57~62, (2021), DOI : 10.7236/JIIBC.2021.21.4.57
[9] Ilango Sriram, Ali Khajeh-Hosseini, "Research Agenda in Cloud Technologies", arXiv:1001.3259 (2010), https://arxiv.org/pdf/1001.3259.pdf
[10] Weisong Shi. George Pallis, Zhiwei Xu, "Edge Computing ", Proceedings of the IEEE, (2019), DOI:10.1109/JPROC.2019.292828
[11] JongPil Youn, JongTae Lim, "Fog and Edge Computing Technology Analysis and Future Prospects from an IoT Perspective", The Journal of KINGComputing 2020, vol.16, no.6, pp. 26-37 (12 pages), (2020)
[12] Alemayehu, Temesgen Seyoum, Cho, We-Duke, "Distributed Edge Computing for DNA-Based Intelligent Services and Applications: A Review", KIPS Transactions on Computer and Communication Systems Volume 9 Issue 12 / Pages.291-306, (2020), DOI:10.3745/KTCCS.2020.9.12.291
[13] Pekka Pääkkönen, Daniel Pakkala, Jussi Kiljander and Roope Sarala, "Architecture for Enabling Edge Inference via Model Transfer from Cloud Domain in a Kubernetes Environment", VTT Technical Research Centre of Finland, 90571 Oulu, Finland, (2020), DOI:10.3390/fi13010005
[14] Dong-Jin Shin et. el, "Big Data-based Sensor Data Processing and Analysis for IoT Environment," The Journal of The Institute of Internet, Broadcasting and Communication(IIBC) Vol. 19, No.1, pp.117-126,(2019), DOI:10.7236/JIIBC.2019.19.1.117
[15] Young-ho Ko, Gyu-Seong Heo, and Sang-Hyun Lee, "A Study on Distributed System Construction and Numerical Calculation Using Raspberry Pi," International Journal of Advanced Smart Convergence Vol.8 No.4 194-199(2019), DOI:10.7236/IJASC.2019.8.4.194
[16] BRENDAN BURNS, BRIAN GRANT, DAVID OPPENHEIMER, ERIC BREWER, AND JOHN WILKES, "Borg, Omega, and Kubernetes", ACM Queue, (2016)
[17] Eunsook Kim, Kyungwoon Lee, Chuck Yoo, "On the Resource Management of Kubernetes", IEEE, 2021 International Conference on Information Networking (ICOIN), (2021), DOI:10.1109/ICOIN50884.2021.9333977
[18] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems 25, (NIPS 2012)
[19] Andrew G, Howard Menglong, Zhu Bo, Chen Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv:1704.04861 (2017)
[20] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks" (2019)
[21] Mahhub Hussain, Jordan J.Bird, Diego R. Faria, "A Study on CNN Transfer Learning for Image Classification", 19th Annual UK Workshop on Computational Intelligence (2018)
[22] Matthias Langer, Zhen He, Wenny Rahayu and Yanbo Xue, "Distributed Training of Deep Learning Models: A Taxonomic Perspective", arXiv:2007.03970 (2020), DOI:10.1109/TPDS.2020.3003307
[23] Dongsuk Yook, Hyowon Lee, and In-Chul Yoo, "A survey on parallel training algorithms for deep neural networks", The Journal of the Acoustical Society of Korea Vol.39, No.6, (2020), DOI:10.7776/ASK.2020.39.6.505