Dyanmic Distribution of Edge Intelligence at the Node Level for Internet of Things.

Hawzhin Mohammed*, Tolulope A. Odetola*, Nan Guo**, Syed Rafay Hasan*

*Department of Electrical and Computer Engineering, Tennessee Tech University, Cookeville, TN, 38505
**Center for Manufacturing Research, Tennessee Tech University, Cookeville, TN, 38505

Abstract—In this paper, dynamic deployment of Convolutional Neural Network (CNN) architecture is proposed utilizing only IoT-level devices. By partitioning and pipelining the CNN, it horizontally distributes the computation load among resource-constrained devices (called as horizontal collaboration), which in turn increases the throughput. Through partitioning, we can decrease the computation and energy consumption on individual IoT devices and increase the throughput without sacrificing accuracy. Also, by processing the data at the generation point, data privacy can be achieved. The results show that throughput can be increased by 1.55x to 1.75x for sharing the CNN into two and three resource-constrained devices, respectively.

Keywords: Internet of Things, Deep Learning, Pipeline, Distribution, Convolutional Neural Network, CNN.

I. INTRODUCTION

1 Internet of Things (IoT) devices are widespread in the smart city, smart grid, advanced manufacturing, health monitoring, and other modern applications. The IoT device combines different sensors, objects, and smart devices that can communicate with each other without human interference [1]. The IoT devices perform autonomously in conjunction with other devices. Cisco Inc. forecasts that 500 billion devices will be connected to the internet by 2030 [2]. Each IoT device collects data through its sensors, communicates with its surroundings, and over the internet with other devices. A network of these connected IoT devices forms the IoT network. Data generated by these smart connected IoT devices used in the analysis to assist make more informed decisions and actions [3].

As the number of IoT devices increases, the number of data collected by them also increases. This data needs to be sent to the data center for analysis and decision-making. Sending an enormous amount of data over the internet cause delay, bandwidth occupation, power consumption, and raise privacy issues. To solve these problems, data need to be processed at the location of the generation [4]. Edge computing is a model which brings data processing and data storage closer to the location where it is generated, to improve response times, bandwidth utilization, energy consumption, and to preserve privacy [5].

Convolutional Neural Network (CNN) in recent times has become a state-of-the-art machine learning technique and has gained adoption in fields like computer vision, natural language processing, object detection among others. The huge computation and memory requirement of CNN makes their deployment on resource-constrained edge devices (e.g. IoT devices) a challenging task [6]. Many research works have proposed different approaches to ensure edge devices accommodate computational and memory requirements of CNN. Traditional techniques include cloud computing which involves the generation of input images from edge devices which are transmitted to the cloud for inference and sent back to edge devices [6] as seen in Fig. 1. This approach has raised privacy concerns with the drawback of high latency [6], [7].

Another approach involves model compression using techniques such as pruning, parameter quantization, Huffman encoding, and hash function. This approach achieves a reduction in computing workloads but leads to a reduction in classification accuracy [8]. The resolution of latency issues has inspired edge computing technologies leading to Edge-based Intelligence (EBI) which alleviates and migrates computation tasks from the cloud to the network edges resulting in a reduction in network communication overhead, lower latency, and a more energy-efficient CNN inference compared to cloud computing [6].

EBI adds some amount of latency between edge devices and edge servers as shown in Fig. 1. EBI is also dependent on the available bandwidth between the edge device and the edge server, as bandwidth decreases, inference’s latency increases especially in volatile bandwidth applications [9]. Moreover, applications such as smart city security, traffic management, or disaster relief in remote and humanly inaccessible areas, require to take decisions more time which is very challenging for resource-constrained devices. Hence, to improve the performance of EBI of CNNs in terms of latency, the inference of CNNs can be done between edge devices excluding edge servers thereby improving latency. Also, the network can be
made dynamically and computation required for intelligence can be distributed among multiple devices (horizontal collaboration).

In this paper, dynamic partitioning methodology (DPM) is explored for the optimal distribution of the layers of CNN among edge devices to achieve faster and accurate inference.

The main contribution of the paper is as follows:

- Dynamic distribution of intelligence among resource-constrained devices.
- Insight on how to optimally divide the CNN among different IoT devices or workers to increase the throughput.

The remainder of this paper is organized as follows: The proposed methodology is presented in Section II. Section III is about how to optimally divide a CNN network. Experimental setup is presented in Section IV. Section V presents the results. Comparison with state-of-the-art is presented in Section VI. Conclusions are drawn in Section VII.

**Algorithm 1 Dynamic partitioning methodology - DPM**

1: **Input to DPM:**
2: Number of MAC operations per layer
3: Number of layers in the CNN
4: Communication channel bandwidth
5: Number of workers
6: **Operation:**
7: Calculate the total MAC operations in the CNN
8: Divide the result by the number of the worker devices
9: Divide the CNN such that each part has approximately the same number of MAC operations
10: Calculate the data size of each part
11: if Output data size > Communication channel bandwidth then
12: Move the division location to decrease the output size
13: end if
14: Distribute the parts among the worker devices
15: **Output:** Optimal layer division and distribution

**II. PROPOSED METHODOLOGY**

In this section, the proposed methodology is explained. First, some terminology that has been used in this paper needs to be defined. An edge device or the IoT device is the device that collects data at the edge of the network. The worker is an edge device that shares its resources with other edge devices (workers). The requester is the edge device that uses other workers’ resources to speed up the inference process. Also, the requester is responsible for creating the network and provide the data (image) for inference. The inputs to DPM are the number of workers, size of data that need to be sent over the communication channel, number of layers, and number of MAC operations per layer (line 2 to 5 in Algorithm I). Algorithm I and Fig. 2 summarize the proposed methodology.

Initially, the algorithm calculates the total number of MAC operations by multiplying the number of layers by the number of MAC operations per layer (line 7). Then, it calculates the number of MAC operations per worker by dividing the total number of MAC operations by the number of workers (line 8). After that, the algorithm divides the CNN such that each worker gets an equal number of MAC operations (line 9). Then, it calculates the data size of each part and compares it with communication channel bandwidth, if the communication channel cannot handle the calculated data size then the algorithm moves the division location to the next layer (lines 10, 11, and 12). Finally, the algorithm distributes parts among workers (line 14).

In this way, the DPM guarantees no channel data overflows no worker computation overflows, and maximizes the throughput. This algorithm also makes sure that each worker can process the data that it receives.

**III. HOW TO OPTIMALLY DIVIDE A CNN**

One important metric of deployment CNN on a hardware unit is the number of multiply-accumulate (MAC) operations. As the number of MAC operations increases the computation process increases and consequently, the time to finish inference of an image increases. For example, LeNet has seven layers [9], as shown in Fig. 3a. Three of these layers are convolution, two of them are max pooling, and the final two are fully connected. Table I shows the number of MAC operations per layer (Table II) shows the advantages of distributed edge intelligence, further discussed in Section V. The convolution layers have higher

| Layer # | Layer type   | MAC operation | Output size |
|---------|--------------|---------------|-------------|
| I       | Convolution  | 86400         | 3456        |
| II      | Max Pooling  | 3460          | 864         |
| III     | Convolution  | 153600        | 1024        |
| IV      | Max Pooling  | 1020          | 256         |
| V       | Convolution  | 30720         | 120         |
| VI      | Fully Connected | 10080     | 84          |
| VII     | Fully Connected | 840        | 10          |

| Case   | # worker | Time/image (ms) | Communication time (ms) |
|--------|----------|-----------------|-------------------------|
| I      | One      | 5.40            | 0                       |
| II     | Two      | 3.48            | 2.13                    |
| III    | Three    | 3.09            | 2.13 and 1.65           |
MAC operations compared to other types of layers [10]. To optimally divide a CNN, the number of MAC operations has to be near equally divided among the number of participated devices, i.e., workers.

But the number of MAC operations for pipeline operations is just half of the story. The other half is the amount of the data (layer output) that needs to be sent over the communication channel. When a CNN runs on a single device, the amount of data that is sent from one layer to the other does not make much difference i.e. minimal time is spent in transferring data between layers. Whereas for pipeline operations where a communication channel involves between the workers, the layer’s output size determines the latency due to the communication, as shown in Figs [3b] and [3c]. The size of the data that needs to be sent from one worker to another depends on the location where the partition of CNN is made. For instance, in Fig. [3a] if the CNN has been divided such that conv1 and pool1 be on one worker and the rest be on another then, in this case, the first worker needs to send 864 of data to the other worker. But, if the CNN has been divided such that conv1, pool1, conv2, and pool2 be on worker one and the rest of the CNN are deployed on other workers, then first worker requires to send 256.

To pipeline the CNN, two metrics must be taken into consideration, the number of MAC operations in each device and the size of the data that needs to be transmitted (as shown in Table II). The algorithm is optimized to ensure that an equal number of operations occur at each worker with minimal data output or data communication size. Choosing minimal communication data to be sent, increases the CNN’s throughput.

\[\text{Pipeline time} \propto \text{Number of MAC operations}\]

\[\text{Pipeline latency time/stage} \propto \text{Size of output}\]

DPM searches for optimal combination among the number of workers, MAC operation, and data communication size, using Algorithm 1, to divide the CNN in the location that guarantees equal MAC distribution with minimal communication data.

### IV. Experimental Setup

In this section, experimental setup is explained that is used to conduct experiments to confirm the effectiveness of DPM.

#### A. Device Type(s)

The testbed consists of Raspberry Pi 3 Model B, which is the third generation of Raspberry Pi. Its specification is: 1) quad-core 1.2GHz Broadcom BCM2837 64bit CPU, 2) 1GB RAM, 3) BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board, and 4) 100 base Ethernet interface. These devices are utilized because they are one of the most commonly used platforms in modern IoT-based network experiments [11]–[13].

There are three scenarios, namely, 1) full LeNet CNN on a single worker device, 2) LeNet CNN is divided between two different worker devices, and 3) LeNet CNN is divided among three different worker devices. As shown in Fig. [3a] in the first scenario there is one case which is the full LeNet running on a single worker device. Table III shows all scenarios and cases.

The second scenario has six cases. The LeNet has been divided at different locations (layer) between the two worker devices. As shown in Table III in the first case the LeNet has been distributed between the two workers, where the first worker runs only the first convolutional layer (conv1) and the rest of the CNN runs on the second worker. In the second case, the first worker runs the first two layers (conv1 and pool1), and the rest of the CNN is runs on the second worker, as shown in Fig. [3b] The partitioning of the CNN continue in this way and finally, in the last case, the first worker runs all the layer of LeNet CNN except for last fully connected layers is handled by the second worker.

In the third scenario, as shown in Table III the first two layers (conv1 and pool1) are run on the first worker, conv2 and pool2 are run on the second and the rest of the layers are run on the third worker. The second case is conv1 and pool1 is run on the first worker, conv2, pool2, and conv3 are run on the second and the rest of the layers are run on the third worker, as shown in Fig. [3c].
TABLE IV: Throughput percentage of one, two, and three workers.

| Case | # Worker | Throughput | Time (ms) |
|------|----------|------------|-----------|
| I    | One      | 100%       | 540.103   |
| II   | Two      | 155%       | 347.780   |
| III  | Three    | 175%       | 308.457   |

V. RESULTS AND DISCUSSIONS

In this section, two types of operations have been discussed, non-pipeline and pipeline operations. In a non-pipeline (case I), there is one worker that contains all the LeNet CNN. In pipeline (case II and case III), the LeNet CNN distributed between the workers to increase the throughput (horizontal collaboration). As it can be seen from Table IV, pipeline cases outperform non-pipeline case. Also, when the number of workers increases, the throughput increases, and the time to inference decreases. For the two workers in Table III and Table IV, DPM chooses the best performance case among the six cases in Table III to compare with the best two cases of three workers.

From Table IV when the LeNet CNN run on one worker (non-pipeline) to classify 100 images, the device took more than 540 ms. To classify the same number of images using pipeline operation, with two workers it took about 348 ms and for three workers the time decreased to about 308 ms. Using a pipeline for inference can increase the throughput up to 155% for two workers and up to 175% for three workers.

For single image inference, as shown in Table II in the non-pipeline case when there is only one worker, to classify one image it took 5.4 ms. The communication time is zero as there is no other device involved in the classification. In case II, the communication time between the two worker nodes to transmit 3456 (as the LeNet CNN divided at pool1) took 2.13 ms. To classify an image, the two workers need an extra 2.13 ms added to the inference time compare to the non-pipeline. For 100 images the communication time is 213 ms added to the 540 ms for the normal inference which becomes 753 ms, but because of the pipelining, the time for inference decreases to 348 ms. These decrements in time are due to the pipelining operation as each worker does its data processing without waiting for the previous worker. The decrements are even more substantial with case III when three workers cooperate to classify the images. Even with the increase of the number of worker nodes which lead to increase in the number of communication link and consequently communication latency, the classification time decreases.

VI. COMPARISON WITH STATE-OF-THE-ART

Zhao et. al [7] proposes distributed intelligent video surveillance using distributed deep learning models. This approach makes use of a multi-layer edge computing platform comprising of monitoring devices, cloud servers, and multilayer edge nodes. This approach does not intimate how to partition the deep learning model based on the available edge nodes. Li et. al [6] introduces a co-inference framework for CNN models to achieve low-latency edge intelligence. This approach proposes the partitioning of CNN models on edge devices and the edge server to achieve inference. This back and forth between the edge servers and edge devices introduces latency.

Mao et. al [8] proposes a local distributed computing approach that utilizes parallel computation for the convolution and fully connected layers of the CNN model. Their approach uses a Biased One-Dimensional Partition scheme for the partitioning of convolution layers. Mao et. al [14] also achieves computation parallelism for the distributed edge computing system. Their approach uses greedy two-dimensional partition to find the optimal partition strategy for splitting layers of the CNN models. Both approaches use a specialized node to schedule and distribute workloads to other edge devices on the network. After completion of operation at every node, the results are sent to the specialized node for re-scheduling and re-distribution. In their approach, the back and forth between the specialized nodes and the other nodes on the network will increase the latency. Operations in this technique are not parallel thereby leading to lower throughput.

VII. CONCLUSION

In this paper, pipeline methodology has been utilized to share the computation and memory requirements for the inference of CNN. This makes the CNN inference capable of deploying on resource-constrained devices without changing the CNN structure or sacrificing accuracy. Also, it increases the throughput and makes it capable of use in real-time applications. By sharing the CNN between two resource constraint devices, the inference can speed up by 1.55x, and by using three resource constraint devices the inference can speed up by 1.75x. To the best of the authors’ knowledge, this is the first work in deploying the CNN inference phase on multiple resource-constrained IoT node devices to achieve higher throughput.
Part of this work is supported by Faculty Research Grant 2020-2021 from the office of research at Tennessee Tech University.

REFERENCES

[1] M. Ambrosin and et. al, “On the feasibility of attribute-based encryption on internet of things devices,” IEEE Micro, vol. 36, pp. 25–35, 2016.
[2] Cisco, “Internet of Things,” At-a-Glance, 2016. [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/ [3] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, “Internet of Things security and forensics: Challenges and opportunities,” 2018.
[4] Z. Zhou and et. al, “Robust mobile crowd sensing: When deep learning meets edge computing,” IEEE Network, vol. 32, no. 4, pp. 54–60, 2018.
[5] C. Ding and D. Tao, “Trunk-branch ensemble convolutional neural networks for video-based face recognition,” IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 4, pp. 1002–1014, 2017.
[6] E. Li, Z. Zhou, and X. Chen, “Edge intelligence: On-demand deep learning model co-inference with device-edge synergy;” in Proceedings of the 2018 Workshop on Mobile Edge Communications. ACM, pp. 31–36.
[7] Z. Zhao, K. M. Barijough, and A. Gerstlauer, “Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2348–2359, 2018.
[8] J. Mao and et. al, “MoDNN: Local Distributed Mobile Computing System for Deep Neural Network,” in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017, pp. 1396–1401.
[9] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner et al., “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.
[10] M. T. Hailesellasie and S. R. Hasan, “MulNet: A Flexible CNN Processor With Higher Resource Utilization Efficiency for Constrained Devices,” IEEE Access, vol. 7, pp. 47 509–47 524, 2019.
[11] O. Hamdan, H. Shanableh, I. Zak, A. Al-Ali, and T. Shanableh, “IoT-based interactive dual mode smart home automation,” in International Conference on Consumer Electronics (ICCE). IEEE, 2019, pp. 1–2.
[12] D. Pavithra and R. Balakrishnan, “IoT based monitoring and control system for home automation,” in global conference on communication technologies (GCCT). IEEE, 2015, pp. 169–173.
[13] V. Vujović and M. Maksimović, “Raspberry Pi as a Sensor Web node for home automation,” Computers & Electrical Engineering, vol. 44, pp. 153–171, 2015.
[14] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, “MeDNN: A Distributed Mobile System with Enhanced Partition and Deployment for Large-Scale DNNs,” in Proceedings of the 30th International Conference on Computer-Aided Design. IEEE Press, 2017, pp. 751–756.