Receptive Field-based Segmentation for Distributed CNN Inference Acceleration in Collaborative Edge Computing

Nan Li, Alexandros Iosifidis and Qi Zhang
DIGIT, Department of Electrical and Computer Engineering, Aarhus University.
Email: {linan, ai, qz}@ece.au.dk

Abstract—This paper studies inference acceleration using distributed convolutional neural networks (CNNs) in collaborative edge computing network. To avoid inference accuracy loss in inference task partitioning, we propose receptive field-based segmentation (RFS). To reduce the computation time and communication overhead, we propose a novel collaborative edge computing using fused-layer parallelization to partition a CNN model into multiple blocks of convolutional layers. In this scheme, the collaborative edge servers (ESs) only need to exchange small fraction of the sub-outputs after computing each fused block. In addition, to find the optimal solution of partitioning a CNN model into multiple blocks, we use dynamic programming, named as dynamic programming for fused-layer parallelization (DPFP). The experimental results show that DPFP can accelerate inference of VGG-16 up to 73% compared with the pre-trained model, which outperforms the existing work MoDNN in all tested scenarios. Moreover, we evaluate the service reliability of DPFP under time-variant channel, which shows that DPFP is an effective solution to ensure high service reliability with strict service deadline.

Index Terms—Distributed CNNs, Receptive field, Edge computing, Service reliability, Time-critical IoT

I. INTRODUCTION

Convolutional neural networks (CNNs) have been widely studied in performing emerging Internet of Things (IoT) applications, such as autonomous driving and augmented reality. However, running computationally intensive CNNs solely at resource-constrained IoT devices is often infeasible to meet stringent deadline in time-critical applications [1]. To solve this issue, various results show that DPFP can accelerate inference of VGG-16 up to 73% compared with the pre-trained model, which outperforms the existing work MoDNN in all tested scenarios. Moreover, we evaluate the service reliability of DPFP under time-variant channel, which shows that DPFP is an effective solution to ensure high service reliability with strict service deadline.

To partition a CNN inference task, segment-based spatial partitioning can be used to partitions an input tensor along the largest dimension, e.g., height or width. Two examples of segment-based spatial partitioning, kernel-size based segmentation and computing-power based segmentation, are proposed to partition an input tensor flexibly depending on the computation resource of each ES and the number of available ESs. We implemented the existing works and tested their inference accuracy, as shown in Table I. It can be seen that the existing works ignore the effect of stride and padding for each CL which can cause intolerable inference accuracy loss. To solve this issue, we propose a receptive field-based segmentation which achieves the exact same result as the pre-trained model, namely without sacrificing any inference accuracy. Note that all methods following the approach used in [1], [7–9] are expected to suffer from similar performance drops. For this reason, hereafter we consider MoDNN as a representative method of this category of methods and conduct extensive comparisons with our proposed RFS which achieves exact
calculation of the model’s output.

To parallelize the sub-tasks among ESs, the fused-layer parallelization (i.e., several CLs are fused into one block, referred to as fused block) was used in the previous works [8]. However, in these works the sub-output tensors of all secondary ESs are entirely sent back to primary ES and merged into a new input tensor, then the primary ES partitions the new input tensor into several sub-input tensors and distributes each of them to individual ES. This will result in substantial communication overhead. To further reduce the communication cost, we propose the collaborative edge computing for fused-layer parallelization (Fig. 1), in which only a small fraction of the sub-output on each ES needs to be exchanged with the other collaborating ESs after computing each fused block. This is because each ES already has most of its input to proceed with the next fused block; therefore, it only needs to obtain the missing information that is needed to proceed with the next fused block from its collaborating ESs.

This paper studies inference acceleration using distributed CNNs in collaborative edge computing network. Our contributions are summarized as follows,

- We propose RFS to partition a CNN inference task among the collaborative ESs without compromising the inference accuracy of the pre-trained model.
- We propose a novel fused-layer parallelization using collaborative edge computing which reduces the computation time and communication overhead. We formulate an optimization problem to minimize the overall inference time and use dynamic programming for fused-layer parallelization (DPFP) to solve this problem.
- We conduct experiments on the high-end GPUs RTX 2080TI and GTX 1080TI, as well as on the embedded GPU JETSON AGX Xavier, to measure the computation time. The experimental results show that our solution can accelerate inference of VGG-16 up to 73% compared with a standalone ES running the pre-trained models. This outperforms the state-of-the-art work MoDNN [1].
- We evaluate the service reliability of DPFP under time-varying channel. It shows that DPFP is effective to ensure high service reliability with strict service deadline.

The remainder of this paper is organized as follows. Section II presents the system model of distributed CNN inference using collaborative ESs. In Section III we propose a novel fused-layer parallelization using collaborative edge computing and formulate the optimizing problem, then in Section IV we use DFDP to solve this problem. The simulation results are presented and discussed in Section V and the conclusions are drawn in Section VI. The source code will be made publicly available at https://gitlab.au.dk/netx/agileiot/dpfp.git.

II. SYSTEM MODEL

This paper studies a collaborative edge computing network, in which an IoT device can offload CNN inference task to a primary ES, which can partition the task and distribute the sub-tasks to multiple secondary ESs and let them collaboratively solve the inference task, as shown in Fig. 1. The inference task is assumed to use a CNN G with N CLs and several fully connected layers (FLs). The connection between the primary ES and the secondary ESs is high-speed Ethernet. The set of ESs can be denoted as $E = \{e_1, \cdots, e_K\}$, where $e_1$ represents the primary ES and the others are secondary ESs. To accelerate a CNN inference task, the primary ES can partition the task into several sub-tasks, each assigned to and processed in parallel on the selected ES, as shown in Fig. 1.

A. How to parallelize CNN computation in ESs

To reduce communication cost after each CL, we use the fused-layer parallelization to partition CNN model $G$ into $M + 1$ fused blocks, where the last fused block is composed of all the FLs ($M \leq N$). In our proposed scheme (Fig. 1), the primary ES is responsible for deciding the data size to be exchanged. The set of the fused blocks is denoted as $F = \{f_1, f_2, \ldots, f_{M+1}\}$, where $f_m$ is the $m$th fused block. We use $l^f_m$ and $r^f_m$ to denote the start layer index and end layer index of fused block $f_m$, $m \neq M + 1$, which subject to

$$l^f_m = \begin{cases} 1, & f = f_1, \\ l^f_{m-1} + 1, & f \in F \setminus \{f_1, f_{M+1}\}. \end{cases}$$

B. RFS based CNN task partitioning

In CNN, every kernel (or filter) looks at a specific part of the input tensor (named as Receptive field), performs multiplication-addition operations, and then moves by a defined number of pixels (stride). Assuming the kernel size, the padding size and the stride size of CL $i$ are $k_i \times k_i$, $p_i$ and $s_i$, respectively, the attributes of CL $i$ can be calculated by [11]:

$$OF_i = [(IF_i + 2p_i - k_i)/s_i] + 1,$$

$$j_i = j_{i-1}s_i,$$

$$r_i = r_{i-1} + (k_i - 1)j_{i-1},$$

$$\sigma_i = \sigma_{i-1} + [(k_i - 1)/2 - p_i]j_{i-1},$$

where $OF_i$ is both the width and height of the output feature of CL $i$, $j_i$ is the cumulative stride (referred to as jump) in the output tensor, $r_i$ is the receptive field size of the output tensor, and $\sigma_i$ is both the row index and column index of the center position of the receptive field of the first output feature. Note that the input tensor of CL $i$ is the output tensor of CL $i - 1$, that means, $IF_i = OF_{i-1}$.

To partition the task efficiently among ESs, taking into account the communication cost for distributing the task and exchanging the intermediate outputs among the ESs, we propose RFS to partition the input tensor into sub-inputs along the largest dimension of the input. An example of partitioning an input tensor into two sub-inputs is shown in Fig. 2.

We define a variable $\eta^f_{e_k}$ to denote the ratio of the sub-output on ES $e_k$ to the entire output of the fused block $f_m$, $m \neq M + 1$ which is subject to

$$\sum_{e_k \in E} \eta^f_{e_k} = 1, \forall f_m \in F \setminus \{f_{M+1}\},$$
where $e_k$ is the number of channels of the input tensor of $f_m$. For $f_m, m \in [2, M]$, ES $e_k$ needs to receive the missing data of its sub-input from its collaborating $e_{k-1}$ and $e_{k+1}$.

The received data size between ES $e_k$ and its collaborating $e_{k-1}$ and $e_{k+1}$, $EX^e_{f_m,k-1}$ and $EX^e_{f_m,k+1}$, are denoted as (13) and (14). After computing $f_M$, the sub-outputs of all the secondary ESs will be sent to the primary ES, which will merge them into the new input for $f_{M+1}$. In general, the exchanged data size for $f_m$, $S(f_m, E)$, can be calculated as (15), where $c^\text{out}_{f_M}$ is the number of channels of the output tensor of $f_M$.

### B. Minimizing total task inference time

During the data exchanging phase, we assume that the data transmission is full-duplex and two ESs can communicate with each other simultaneously. With 100 Gbps Ethernet, the communication time for exchanging data is very small compared with the computation time, therefore, the performance of our proposed scheme will be affected marginally, even if it does not use full-duplex transmission mode. Assuming the transmission rate between ESs is $\alpha$, the communication time for exchanging data between ESs for $f_m$ can be denoted as

$$T_{\text{com}}(f_m, E) = S(f_m, E)/\alpha.$$

### A. Exchanged data size between ESs

For a tensor $(\text{height}, \text{width}, \text{channel})$ with type in float32, its size is calculated as $4 \cdot \text{height} \cdot \text{width} \cdot \text{channel}$ in bytes.

Before starting the computation of the first fused block $f_1$, the primary ES sends sub-inputs to the secondary ESs. The exchanged data size of $f_1$ is calculated as

$$S(f_1, E) = \sum_{e_k \in E \setminus \{e_1\}} 4 \left( |I_{f_1}^{e_k} - IS_{f_1}^{e_k}| + 1 \right) IF_{f_1} c^\text{in}_{f_1},$$

where $c^\text{in}_{f_1}$ is the number of channels of the input tensor of $f_1$. For $f_m, m \in [2, M]$, ES $e_k$ needs to receive the missing data of its sub-input from its collaborating $e_{k-1}$ and $e_{k+1}$.

The received data size between ES $e_k$ and its collaborating $e_{k-1}$ and $e_{k+1}$, $EX^e_{f_m,k-1}$ and $EX^e_{f_m,k+1}$, are denoted as (13) and (14). After computing $f_M$, the sub-outputs of all the secondary ESs will be sent to the primary ES, which will merge them into the new input for $f_{M+1}$. In general, the exchanged data size for $f_m$, $S(f_m, E)$, can be calculated as (15), where $c^\text{out}_{f_M}$ is the number of channels of the output tensor of $f_M$.

### III. PROBLEM FORMULATION

To effectively minimize the overall inference time of a CNN inference task, the key is to trade off the communication and computation time used in processing CLs. This is because increasing number of ESs can in general reduce the computation time; however, it can result in more communication overhead due to more exchanged data among ESs. Similarly, increasing the number of fused blocks can reduce computation time while increasing exchanged data size consequently increasing communication time. This section firstly calculates the exchanged data size between ESs, then formulates the joint optimization problem to minimize the overall inference time.

$$EX^e_{f_m,k-1} = 4 \max \left( |O_{f_m}^{e_{k-1}} - IS_{f_m}^{e_k}|, 0 \right) + 1 \left[ IF_{f_m} c^\text{in}_{f_m}, f_m \in F \setminus \{f_1, f_{M+1}\}, \right.$$

$$EX^e_{f_m,k+1} = 4 \max \left( |I_{f_m}^{e_k} - OS_{f_m}^{e_{k+1}}, 0 \right) + 1 \left[ IF_{f_m} c^\text{in}_{f_m}, f_m \in F \setminus \{f_1, f_{M+1}\}, \right.$$

$$S(f_m, E) = \begin{cases} 
\sum_{e_k \in E \setminus \{e_1\}} 4 \left( |I_{f_m}^{e_k} - IS_{f_m}^{e_k}| + 1 \right) IF_{f_m} c^\text{in}_{f_m}, & f_m = f_1, \\
\sum_{e_k \in E \setminus \{e_1\}} \sum_{k \in E} EX^e_{f_m,k-1} + EX^e_{f_m,k+1}, & f_m \in F \setminus \{f_1, f_{M+1}\}, \\
\sum_{e_k \in E \setminus \{e_1\}} 4 \left( |O_{f_M}^{e_k} - OS_{f_M}^{e_k}| + 1 \right) OF_{f_M} c^\text{out}_{f_M}, & f_m = f_{M+1}.
\end{cases}$$
During the computing phase, the primary ES needs to wait until the last collaborating ES completes the computation. Assuming the computation time of ES $e_k$ for the fused block $f_m$ is $t_{f_m}^e$, the computation time for $f_m$ can be denoted as

$$T_{\text{comp}}(f_m, E) = \begin{cases} \max \left(t_{f_m}^e, \ldots, t_{f_m}^{K_m}\right), & f_m \neq f_{M+1}, \\ t_{f_m}^e, & f_m = f_{M+1}. \end{cases} \quad (17)$$

Note that $f_{M+1}$ is solely computed at primary ES $e_1$. Hence, the total inference time for $f_m$ can be expressed as

$$T_{\text{inf}}(f_m, E) = T_{\text{comp}}(f_m, E) + T_{\text{com}}(f_m, E). \quad (18)$$

The objective to minimize the overall inference time of the inference task is formulated as follows:

$$\min_{E,F} \sum_{f_m \in F} T_{\text{inf}}(f_m, E), \quad (19)$$

subject to (1), (6) and (7).

IV. DYNAMIC PROGRAMMING FOR FUSED-LAYER PARALLELIZATION

The optimization goal of (19) is to search the optimal number of ESs and the optimal partition of a CNN model. Therefore, this minimization problem can be divided into two steps: (i) to determine optimal number of the fused blocks of the CNN model for the given number of ESs; (ii) to determine the optimal number of ESs. Assuming the selected ESs $E'$ and the ratio $\eta_{f_m}^m$ are given, $T_{\text{comp}}(f_{M+1}, E')$ and the ratio $\eta_{f_m}^m$ are determined no matter how to partition the CNN model. Therefore, the minimization problem (19) is equivalent to minimize the completion time of all the CLs as

$$\min_{F} \sum_{f_m \in F, f_m \neq M+1} T_{\text{inf}}(f_m, E'). \quad (20)$$

For ease of description, the inference time $T_{\text{inf}}(f_m, E')$ of the fused block $f_m$, $m \neq M+1$ can be simplified as $t\left(l_{f_m}^m, l_{f_m}^e\right)$. Assuming the optimal partitioning of $G$ is $F^*$ with $M^* + 1$ fused blocks, the minimal inference time of all the CLs in (20) is equal to the sum of the inference time of each fused block, which can be denoted as

$$t^*(1, N) = \sum_{f_m \in F^* \setminus f_{M+1}} t\left(l_{f_m}^m, l_{f_m}^e\right). \quad (21)$$

This optimization problem can be solved by the well-known rod-cutting problem in dynamic programming [12]. More generally, the optimal inference time can be expressed as

$$t^*(1, N) = \min\{t(1,1), t^*(1, 2) + t^*(3, N), t^*(1, 3) + t^*(4, N), \ldots, t^*(1, N - 1) + t(N, N)\}. \quad (22)$$

Simplifying (22), the minimal inference time for $f_m$ can be expressed as (23), below. Based on dynamic programming, the optimal set for the fused blocks can be obtained as described in Algorithm 1. After the optimal fused blocks $F^*$ are determined for the selected ESs $E'$, the next step is to derive the optimal ESs by comparing the inference time under different number of the selected ESs. The one that has the lowest inference time is the optimal selection of ESs $E'$. To compare the performance of the proposed collaborative edge computing for distributed CNN with the conventional approach, we define the speedup ratio $\rho$ for the optimal $E'$ as

$$\rho = 1 - \frac{T_{\text{inf}}(E')}{T_{\text{pre}}}. \quad (24)$$

where $T_{\text{inf}}(E')$ is the inference time for ESs $E'$, $T_{\text{pre}}$ is the inference time of the pre-trained model running on standalone ES. It is clear that the faster the proposed approach is, the higher speedup ratio will be achieved.

Algorithm 1: Dynamic Programming for fused-layer parallelization (DPFP)

**Input:** CNN model $G$, selected ESs $E'$, Optimal CNN model partitioning point set $S$

**Output:** Optimal Fused blocks $F^*$

**Step 1:** Initialize $t^*(\cdot) \leftarrow \text{MAX}$, $S \leftarrow \emptyset$

**Step 2:** Define a function DPFP $(i,j)$:

function DPFP $(i,j)$:

if $j = i$ then

return $t^*(i, j) \leftarrow t(i, j)$ and $S \leftarrow S \cup j$;

else

let $t_{min} = \text{MAX}$

for $k$ range in $[1, j - i - 1]$ do

$t = t(i, i+k) + \text{DPFP} (i+k+1, j)$

if $t < t_{min}$ then

$t_{min} = t$ and the optimal partitioning point

$t_{opt} = i+k$;

end

end

return $t^*(i, j) \leftarrow t_{min}$ and $S \leftarrow S \cup t_{opt}$;

end

**Step 3:** Using step 2 to calculate $t^*(1, N)$ and $S$, then $F^*$ can be computed according to $S$.

$$t^*(l_{f_m}^m, l_{f_m}^e) = \begin{cases} \min_{1 \leq i \leq l_{f_m}^m - l_{f_m}^e - 1} \left(t\left(l_{f_m}^m, l_{f_m}^e + i\right) + t^*(l_{f_m}^m + i, l_{f_m}^e)\right), & l_{f_m}^e < l_{f_m}^e \leq N. \quad (23) \end{cases}$$
Fig. 3. Speedup ratio vs. No. of ESs at 100Gbps between ESs.

TABLE II
INFEREN CE TIME FOR VARIABLE ESs AT 100 Gbps BETWEEN ESs (ms)

|          | 2 ESs |          | 7 ESs |
|----------|-------|----------|-------|
|          | T^cmp | T^com   | T^cmp | T^com |
| DPFP_RTX 2080TI | 2.26  | 0.08    | 2.34  | 0.35  | 1.67  |
| DPFP_GTX 1080TI | 2.79  | 0.08    | 2.87  | 0.35  | 1.88  |
| DPFP_AGX Xavier | 16.69 | 0.10    | 16.79 | 0.52  | 8.72  |

TABLE III
INFEREN CE TIME FOR 7 ESs WITH VARIABLE TRANSMISSION RATE (ms)

|          | 40Gbps |          | 100Gbps |
|----------|--------|----------|---------|
|          | T^cmp | T^com   | T^inf  |
|          |        |         |        |
| DPFP_RTX 2080TI | 1.38  | 0.74    | 2.12   |
| DPFP_GTX 1080TI | 1.59  | 0.74    | 2.35   |
| DPFP_AGX Xavier | 8.24  | 1.30    | 9.54   |
| MoDNN_RTX 2080TI | 1.01  | 9.97    | 10.98  |
| MoDNN_GTX 1080TI | 1.12  | 9.97    | 11.09  |
| MoDNN_AGX Xavier | 8.23  | 9.97    | 17.20  |

Fig. 4. Speedup ratio vs. variable transmission rate when 7 ESs collaborate.

V. PERFORMANCE EVALUATION

A. Simulation Setup and Methodology

In this simulation, the edge computing network consists of one IoT device and 10 ESs are of equal computing capacity. The data communication between ESs via Ethernet with transmission rate α ranging from 40Gbps to 100Gbps. We use the image recognition model VGG-16 in the experiments. We compare the proposed approach with a state-of-the-art work MoDNN. To provide a realistic performance evaluation, we measured computation time on three different GPU platforms, i.e., RTX 2080TI with 13.45 TFLOPS of float32, GTX 1080TI with 11.3 TFLOPS of float32, and JETSON AGX Xavier with 1.41 TFLOPS of float32. The shape of the input images is 224 × 224 × 3, and the performance metric is speedup ratio ρ.

B. Impact of ESs

TABLE II shows the computation time and communication time with different number of ESs at the transmission rate equal to 100Gbps. Generally, it can be seen that as the number of ESs increases, the computation time decreases, on the contrary, the communication time between ESs increases. This is because when more ESs are sharing the inference task, it will naturally reduce the sub-task size on each ES; however, the total exchanged data size between ESs will increase. When the number of ESs is small, the computation time takes up most of the overall inference time; therefore, the speedup ratio will increase as the number of ESs increases, as shown in Fig. 3. When the number of ESs is larger than 7, the reduced computation time by using more ESs is rather small and is even close to the introduced extra communication time. Therefore, the speedup ratio reaches a plateau.

Comparing the performance of different platforms, it can be seen that, the optimal sets of the fused blocks for GTX 1080TI and RTX 2080TI are almost the same, however, for JETSON AGX Xavier the model is in fact partitioned at every CL. This is because to strike a good balance between the communication time and computation time, it is worth fusing several CLs into one fused block when an ES has high computation capacity, e.g., for the cases of GTX 1080TI and RTX 2080TI; however, for low computation capacity platform, e.g., JETSON AGX Xavier, it does not pay off to fuse several CLs into one fused block. In other words, the extra computation time caused by fused block is more than the saved communication time for low computation capacity platforms.

C. Impact of transmission rate between ESs

TABLE III shows the inference time at different transmission rates between ESs for the case of 7 ESs collaborating. We can see DPFP partitions the model at each CL for JETSON AGX Xavier as MoDNN, whereas for the other two platforms DPFP fuses several CLs into one block. Since computation capacities of the RTX 2080TI and GTX 1080 TI do not have a huge difference, the optimal partitioning solution is the same for these two platforms. Moreover, the partitioning solution varies under different transmission rates in the case of RTX 2080TI and GTX 1080TI. For example, we can observe that the exchanged data increases from 3.7 MBytes at 40 Gbps to 4.375 MBytes at 100 Gbps, which indicates DPFP partitions the model into smaller blocks at a higher transmission rate, to trade off the computation time and communication time. As MoDNN needs to exchange data for merging sub-outputs after every CL and DPFP only exchanges the overlapped data between ESs after each fused block, DPFP has much less communication overhead and can reduce the communication time about 90% at a given transmission rate.

Therefore, as shown in Fig. 4 using the same platform DPFP has better speedup ratio than MoDNN. As the transmission rate increases, the speedup ratio of a platform using DPFP will improve and then reach a plateau. This is because with increasing transmission rate, the communication time gradually decreases and takes up less and less time of the overall inference time, while the computation time nearly
For an inference task, the total task completion time can be processing and minimizes resource demands on IoT device. The inference task and how to distribute the sub-tasks to the device, the primary ES makes decision on how to partition the platform with higher computing capacity uses less computing costs, we propose a novel fused-layer parallelization into fused blocks. In addition, we use dynamic programming to effectively solve the optimization problem of minimizing total computation time. The experimental results show that DPFP can accelerate inference of VGG-16 up to 73% compared with the pre-trained model, which outperforms the state-of-the-art approach MoDNN. Furthermore, we evaluate the service reliability of DPFP under time-variant channel, which shows that DPFP is an effective solution to ensure high service reliability with strict service deadline.

D. Evaluation of service reliability

Instead of offloading the sub-tasks to ESs directly from IoT device, the primary ES makes decision on how to partition the inference task and how to distribute the sub-tasks to the secondary ESs. This full offloading scheme simplifies the processing and minimizes resource demands on IoT device. For an inference task, the total task completion time can be denoted as \( T = T_{\text{off}} + T_{\text{inf}} \), where \( T_{\text{off}} \) is offloading time of an inference task transmitted from IoT device to primary ES.

To ensure real-time inference, we can decide the minimal transmission data rate between IoT device and the primary ES according to Section V-B. For example, to ensure system throughput (e.g., 30 FPS) for an input image (e.g., 125 KBytes), the transmission data rate between IoT device and the primary ES should not be lower than 32 Mbps. Generally, the channel state is stochastic, which may cause fluctuations in offloading time \( T_{\text{off}} \). Assume the offloading time \( T_{\text{off}} \sim N(\mu, \delta^2) \), we can estimate the fluctuation of transmission rate, \( \phi \), based on the three-sigma rule of thumb. In mission critical IoT services, the required service reliability (the probability that inference feedback meets the service deadline) can be in the range of 99% to 99.999%.

TABLE IV shows the service reliability under the time-variant channel of different transmission rate. It can be seen that the service reliability will decrease as the fluctuation of channel status increases. To address this issue, a higher transmission rate can contribute to improving the service reliability. However, relying solely on improving transmission rate cannot guarantee the high service reliability of 99.999%. For example, a standalone ES can achieve max 99.955% service reliability for the scenario at average data rate of 100 Mbps and fluctuation \( \phi = 60.0 \text{ Mbps} \). Note that in a mobile network, large fluctuation in data rate is likely to occur, for example, the uplink data rate of a moving vehicle. In this case, DPFP can still ensure high service reliability of 99.999% and even further.

VI. CONCLUSION

This paper studies inference acceleration through distributed CNN using collaborative edge computing for time-critical IoT applications. We propose RFS to effectively partition an inference task without compromising the inference accuracy of the pre-trained model. To minimize the communication and computing costs, we propose a novel fused-layer parallelization using collaborative edge computing to partition a CNN model into fused blocks. In addition, we use dynamic programming to effectively solve the optimization problem of minimizing total computation time. The experimental results show that DPFP can accelerate inference of VGG-16 up to 73% compared with the pre-trained model, which outperforms the state-of-the-art approach MoDNN. Furthermore, we evaluate the service reliability of DPFP under time-variant channel, which shows that DPFP is an effective solution to ensure high service reliability with strict service deadline.

ACKNOWLEDGMENT

This work is supported by Agile-IoT project (Grant No. 9131-00119B) granted by the Danish Council for Independent Research.

REFERENCES

[1] J. Mao, X. Chen et al., “Modnn: Local distributed mobile computing system for deep neural network,” in Design, Automation Test in Europe Conference Exhibition (DATE) 2017, pp. 1396–1401.
[2] D. Xu, T. Li et al., “Edge Intelligence: Architectures, Challenges, and Applications,” arXiv e-prints, Mar. 2020.
[3] J. Liu and Q. Zhang, “To improve service reliability for ai-powered time-critical services using imperfect transmission in mec: An experimental study,” IEEE Internet of Things Journal, vol. 7, no. 10, pp. 9357–9371, 2020.
[4] Q. Zhang and F. H. P. Fitzek, “Mission critical iot communication in 5g,” in Future Access Enablers for Ubiquitous and Intelligent Infrastructures. Cham: Springer International Publishing, 2015, pp. 35–41.
[5] N. Li, J.-B. Wang, J.-Y. Wang, M. Cheng, and M. Chen, “Online learning based transmission scheduling over a fading channel with imperfect csi,” in GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–6.
[6] J. Liu and Q. Zhang, “Using imperfect transmission in mec offloading to improve service reliability of time-critical computer vision applications,” IEEE Access, vol. 8, pp. 107364–107572, 2020.
[7] C. Hsu, C. Yang et al., “Cooperative convolutional neural network deployment over mobile networks,” in IEEE ICC 2020, pp. 1–7.
[8] C. Zhou, M. H. Samavatian, A. Basha, S. Majumdar, and R. Teodorescu, “Adaptive parallel execution of deep neural networks on heterogeneous edge devices,” in ACM/IEEE SEC. 2019, pp. 195–208.
[9] T. Mohammed, C. Joe-Wong et al., “Distributed inference acceleration with adaptive dnn partitioning and offloading,” in IEEE INFOCOM 2020, pp. 854–863.
[10] W. Luo, Y. Li et al., “Understanding the effective receptive field in deep convolutional neural networks,” in NIPS’16, pp. 4905–4913.
[11] N. M. Zivkovic. Deep Learning for Programmers, 2020. [Online]. Available: https://rubikscode.net/2020/05/18/receptive-field-arithmetic-for-convolutional-neural-networks/.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Third Edition. The MIT Press, 2009.
[13] “IEEE standard for ethernet-amendment 11: Physical layers and management parameters for 100 gbs and 400 gbs operation over single-mode fiber at 100 gbs per wavelength,” IEEE Std 802.3cu-2021, pp. 1–87.
[14] K. Simonian and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in ICLR 2015, May 7–9.