Efficient Computer Vision on Edge Devices with Pipeline-Parallel Hierarchical Neural Networks

Abhinav Goel, Caleb Tung, Xiao Hu, George K. Thiruvathukal*, James C. Davis, Yung-Hsiang Lu
Purdue University, School of Electrical and Computer Engineering, West Lafayette, IN, USA
*Loyola University Chicago, Department of Computer Science, Chicago, IL, USA

***Abstract*—Computer vision on low-power edge devices enables applications including search-and-rescue and security. State-of-the-art computer vision algorithms, such as Deep Neural Networks (DNNs), are too large for inference on low-power edge devices. To improve efficiency, some existing approaches parallelize DNN inference across multiple edge devices. However, these techniques introduce significant communication and synchronization overheads or are unable to balance workloads across devices. This paper demonstrates that the hierarchical DNN architecture is well suited for parallel processing on multiple edge devices. We design a novel method that creates a parallel inference pipeline for computer vision problems that use hierarchical DNNs. The method balances loads across the collaborating devices and reduces communication costs to facilitate the processing of multiple video frames simultaneously with higher throughput. Our experiments consider a representative computer vision problem where image recognition is performed on each video frame, running on multiple Raspberry Pi 4Bs. With four collaborating low-power edge devices, our approach achieves 3.21× higher throughput, 68% less energy consumption per device per frame, and a 58% decrease in memory when compared with existing single-device hierarchical DNNs.**

***Index Terms*—Parallel edge computing, hierarchical DNNs.**

I. Introduction

Deep Neural Networks (DNNs) are the state-of-the-art techniques to perform computer vision tasks on video streams. Because of the significant energy and computation resource requirements of DNNs, video stream processing is usually performed on the Cloud [1]. However, applications with strict throughput, privacy, or network bandwidth constraints must be handled locally [2]. Increasing the efficiency of DNNs will enable more low-power edge devices to process visual data without offloading.

Existing efforts to increase DNN efficiency are largely focused on single-device inference [3, 4]. However, low-power edge devices are commonly deployed in a network, e.g. to enable monitoring of multiple angles of a traffic intersection or a construction site [5]. If these networks have spare computing resources, then *parallel* inference would allow the devices to share resources for faster data processing [6, 7]. For example, if an edge device is not powerful enough to provide the required response time, the device could partition the DNN, and transmit the partitioned tasks to other devices [8].

Some existing works perform parallel DNN inference on edge devices. These methods can be classified as (a) Data parallelism: each collaborating edge device processes a subset of the inputs with the assumption that each device can run the entire DNN [7]; (b) Model parallelism: each DNN layer is partitioned across multiple devices, but requires extensive inter-device communication to map inputs and reduce outputs [12]; (c) Pipeline parallelism: the DNN is partitioned into sets of consecutive layers. Each set is run on a different device; after one device processes a frame, the frame is passed onto another device. This allows the first device to process the next frame for improved throughput [13]. Pipeline parallelism is most suitable for improving the throughput of video stream processing but is currently limited because conventional DNNs have a large variance in resource requirements across layers [14].

We observe that the recent hierarchical DNN architecture [10, 9] is well suited for pipeline parallelism. This architecture is depicted in Fig. 1(a) and detailed in Section II. The small DNNs of the hierarchy can be partitioned to run independently on collaborating devices without a large cross-device resource variance, as seen in Fig. 1(b). Using this insight, this paper proposes a novel technique to perform pipeline-parallel inference of hierarchical DNNs. Our method partitions hierarchical DNNs in a way that balances workloads and reduces communication costs. We show that, when partitioning the hierarchy, it is advantageous to consider the hierarchy structure and the processing time of each DNN.

To evaluate this approach, our experiments compare the video stream processing performance of the proposed method with state-of-the-art techniques [4, 7, 8, 13, 9] in terms of frames per second (FPS), latency, memory and numbers of operations, and energy consumption per device. We vary the hierarchical DNN structures, input resolutions, video lengths, and the number of devices to show that the proposed technique improves throughput for different types of workloads. These experiments are performed using standard computer vision

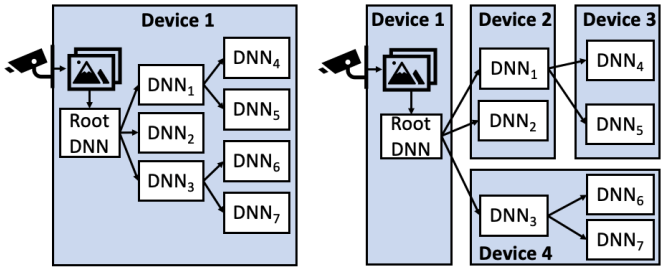


Fig. 1: (a) Existing hierarchical DNNs: multiple small DNNs in the form of a tree. All DNNs along a single root-leaf path process a video frame before the next frame can be processed [9, 10, 11]. (b) Our method: after the root DNN at device 1 processes a frame, the frame is passed onto another device. This allows device 1 to start processing the next frame and creates an inference pipeline to increase throughput.

datasets. We observe a 3.21× increase in FPS, and 60%, 58%, 68% decrease in operations, memory, and energy requirements, respectively. These gains are achieved when using four collaborating Raspberry Pi 4Bs connected via Ethernet.

II. Background and Related Work

A. Hierarchical Deep Neural Networks

Hierarchical DNNs use multiple small DNNs in the form of a tree, as seen in Fig. 1(a) [9, 10, 11, 15]. Each small DNN specializes in an intermediate classification between groups of similar categories. In each level of the hierarchy, a small DNN uses the activation map of its parent and makes an intermediate classification into progressively smaller groups, until a leaf DNN provides the final output (e.g. DNN_4 in Fig. 1). Existing techniques consider the training [10], design [2, 9], or applications [11] of hierarchical DNNs on one device.

Hierarchical DNNs offer an energy-accuracy tradeoff. Since each input is only processed by the small DNNs along one path from the root to a leaf, hierarchical DNNs perform inference more efficiently than conventional DNNs [2, 9], reducing energy consumption by ∼50%. However, they also decrease accuracy by ∼4%, because errors propagate from parent to child DNN. As this tradeoff may be acceptable in practice [9], we investigate improving hierarchical DNN processing throughput by performing pipeline-parallel inference.

B. Three Categories of Parallel Edge Computing

Data parallelism: The input data is partitioned and processed independently by the collaborating devices. Splitting a single input frame impacts the spatial locality of objects and lowers DNN accuracy [16]. Most data-parallel techniques distribute activation maps to perform convolution operations in parallel [14, 7]. MoDNN [7] divides each activation map into overlapping grid cells. MeDNN [14] distributes the activation channels (instead of grids) across devices to avoid repeated operations. All data parallelism techniques assume that each edge device has the capacity to run the entire DNN [17].

Model Parallelism: Each DNN convolution operation is independent of all other operations in the same layer. Model parallelism uses this intra-layer independence to split a DNN into disjoint subsets on multiple devices [12]. Model parallelism techniques usually have significant overhead in communicating activation maps [18]. These methods also suffer from the straggler effect when workloads are imperfectly balanced [16, 18]. Bhardwaj et al. [17] reduce the overhead of model parallelism, but their technique is applicable only when the number of devices is fixed and is known at training time.

TABLE I: Comparison of the proposed method with existing methods. H-DNN: Hierarchical DNN.

Technique	H-DNN	Parallelism Type	Load Balance	Comm. Efficient
Howard et al. [4]	×	None	-	-
Mao et al. [7]	×	Data	✓	×
Zhang et al. [8]	×	Pipeline	×	✓
Hadidi et al. [13]	×	Pipeline + Model	✓	×
Goel et al. [9]	✓	None	-	-
Our Method	✓	Pipeline	✓	✓

Pipeline Parallelism: The DNN is divided into sets of consecutive layers, and each set is deployed on a collaborating device to improve throughput [8, 19]. Conventional DNNs (e.g. VGG and ResNet) have been found to be ill-suited for pipeline parallelism because there is a large variance in resource requirements and communication costs across layers; this results in imbalanced workloads [8]. Zhang et al. [8] show that the inference time of the largest fully-connected layer of VGG-16 is $\sim 15\times$ larger than the inference time of the smallest convolution layer on an edge-class device. To alleviate this issue at the cost of additional overhead, Hadidi et al. [13] combine pipeline parallelism with model parallelism to prevent bottlenecks.

Table I presents properties of existing techniques. The existing parallel techniques either balance loads or reduce communication costs [7, 8, 13]. We propose the first method to perform parallel inference of hierarchical DNNs. This method performs efficient pipeline parallelism by balancing loads and reducing overhead. This work enables the use of pipeline parallelism for improving the throughput in application contexts where multiple edge devices operate in close proximity (e.g. airports, traffic intersections, construction sites, etc.).

C. Our Contributions

As summarized by Table I: (1) This is the first method to perform parallel hierarchical DNN inference to accelerate video stream processing on low-power embedded devices. (2) We develop a mathematical model to estimate the throughput gains with pipelined hierarchical DNNs in different application scenarios. (3) Using this model we present a novel technique that partitions the hierarchical DNN for maximizing throughput with pipeline parallelism. (4) We experimentally measure the factors that impact the throughput of the pipelined hierarchical DNNs.

III. Pipelining Hierarchical Neural Networks

This section describes our pipeline parallelism scheme with hierarchical DNNs. To create a hierarchical DNN inference pipeline, we first identify the factors that impact the processing time of pipeline-parallel hierarchical DNNs and create a model to estimate the throughput with our method (Section III-A). We then use this model to find the hierarchy partition that maximizes the throughput (Section III-B).

A. Throughput of Pipeline-Parallel Hierarchical DNNs

In pipeline-parallel systems, the throughput depends on (1) the number of pipeline stages, (2) the time taken to process each stage, and (3) the communication overhead. The processing time for F frames is given by the general equation: $P_{time} = [(F + \textit{\#pipeline stages} - 1) \times (\textit{max stage processing time})] + \textit{communication time}$ [20]. The steady-state throughput for a video stream is given by $\frac{F}{P_{time}}$. In the proposed technique, the hierarchy structure (depth, number of edges, etc.) and the method used to partition the hierarchical DNN to run on collaborating devices impact the throughput. These factors impact the number of pipeline stages, the processing time, and the overhead.

We use seven parameters to model the throughput of the proposed pipelined hierarchical DNNs. These parameters are

TABLE II: Symbols reference. *: Values are obtained after the hierarchical DNN has been partitioned.

Symbol	Definition
N	Number of collaborating devices
F	Number of frames
Λ	Avg. DNN processing time
τ	Avg. communication time between devices
K	Maximum hierarchical DNN depth
H^*	Avg. number of edge cuts from root to leaf
M^*	Avg. number of DNNs running sequentially on one device

listed in Table II. The average DNN processing time, Λ, depends on each DNN's processing time and rate of use. DNNs may have different rates of use, dependent on two factors. First, the hierarchy structure affects the rate of use; the root DNN is used most often because it processes every video frame, while leaf DNNs (e.g. DNN_7 in Fig. 1) process only a small subset of the frames. Second, a DNN's rate of use is application-dependent; e.g. in an airport, people are more common than cats, and so the DNNs responsible for processing people will be used more often than those for cats. In the same way, the average communication time, τ, depends on the amount of data transferred in each hierarchy edge and the rate at which the edges are used. Each hierarchy edge between a parent and child is used at the same rate as the child DNN (the edge is used only when the child is used).

In pipelined hierarchical DNNs, the hierarchy depth, K, determines the number of stages in the pipeline. When the pipeline is full, a DNN at every level of the hierarchy is processing a frame. $(F + K - 1) \times \Lambda$ is the DNN processing time for F frames when $N = K$ (all stages can run in parallel). When $N < K$, only N DNNs can run in parallel. The hierarchy partition algorithm assigns DNNs to the collaborating devices. If multiple DNNs along one root-leaf path are assigned to the same device, then the DNNs run sequentially on the device. In Fig. 1(b), when Device 4 runs DNN_6, DNN_3 must wait before it can process the next video frame. If M is the average number of DNNs that run sequentially on a single device, then the total DNN processing time can be approximated as $(F + N - 1) \times (M \times \Lambda)$.

After hierarchical DNNs are partitioned to run on collaborating devices, a hierarchy edge is considered to be *cut* if it spans partitions (or devices). When an edge cut is encountered, the DNN activation map needs to be communicated between devices. Thus, for each frame $H \times \tau$ is the communication overhead. The total communication time is $F \times (H \times \tau)$.

Similar to P_{time}, the total time taken to process F frames is approximately $\Big((F+N-1)\times(M\times\Lambda)\Big)+\Big(F\times(H\times\tau)\Big)$. The estimated throughput, $T_{th.}$, of our method is given in eqt. (1).

$$T_{th.} \approx \frac{F}{\Big((F + N - 1) \times (M \times \Lambda)\Big) + \Big(F \times (H \times \tau)\Big)} \quad (1)$$

This model estimates the throughput for different hierarchy structures, devices, and communication media. A hierarchical DNN partition method is required to assign DNNs to devices. The partition must find a tradeoff between the workload size $(M \times \Lambda)$ and the overhead $(H \times \tau)$ to maximize $T_{th.}$.

Model Assumptions: This model operates under the assumption that there is no temporal relationship between frames. For example, eqt. (1) may not accurately estimate the throughput for a video where all frames containing cats appear first, followed by all the frames containing trucks, and so on. We embed this assumption into our model by using averaged values in H and M. This assumption does not sacrifice generality for edge applications because different objects may appear at any time; e.g. over a day traffic cameras see cars, bikes, etc. Furthermore, our analysis only considers the situation when all devices have the same hardware. This assumption suits edge-contexts where homogeneous edge devices are deployed to simplify device management [21].

B. Partitioning Hierarchical DNNs for Pipeline-Parallelism

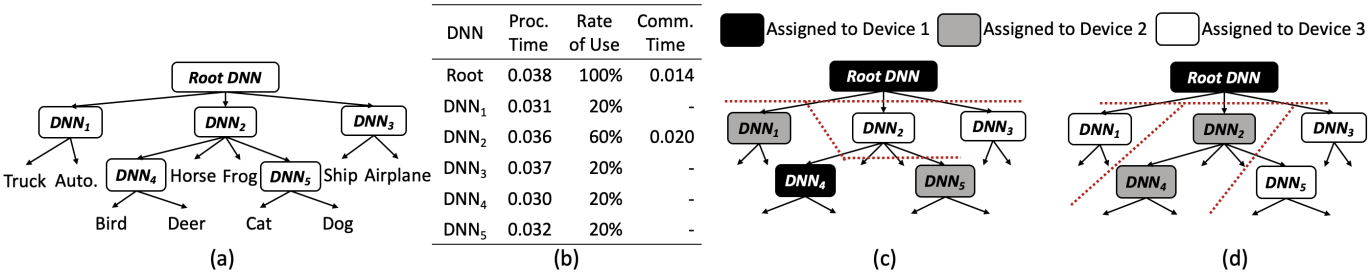
In this subsection, using examples in Fig. 2, we first show how the partition method impacts the throughput. We then describe our novel technique to find a hierarchy partition that maximizes the pipeline-parallel throughput.

1) Impact of Hierarchical DNN Partitions on Throughput: Hierarchical DNNs contain small DNNs in the form of a hierarchy. To perform pipeline-parallel inference with hierarchical DNNs, first, the hierarchy must be partitioned. The hierarchy partition controls the values of H and M in eqt. (1). Each partition is assigned to and run on a collaborating device. Hierarchies can be partitioned in different ways.

To understand how the hierarchical DNN partition method impacts the DNN processing time and communication overhead, consider the example in Fig. 2 when there are three collaborating devices. Fig. 2 (a, b) depict the hierarchical DNN constructed for performing image recognition on images from the CIFAR-10 dataset along with the time taken to process images, the rate of use, the communication time for each DNN in the hierarchy. In this example, DNN_2 requires 0.036 seconds to process a frame and 0.020 seconds to communicate its activation map. Since the costs are comparable, finding hierarchy partitions that reduce the communication overhead is important for obtaining higher throughput. Without loss of generality, in this example, we assume that all the object categories in the dataset are equally probable to appear. Thus, because there are 10 leaves in the hierarchy and 6 of those leaves are rooted at DNN_2, the rate of use for DNN_2 is $\frac{6}{10}$. In other words, 60% of the input frames will be processed by DNN_2. If objects are not equally probable (e.g. in an airport), the data must be sampled [22] to find the rate of use for each DNN before utilizing our approach.

If only the DNN processing times in Fig. 2 are balanced, then $\langle$root, $DNN_4\rangle$, $\langle DNN_1, DNN_3\rangle$, and $\langle DNN_2, DNN_5\rangle$ are assigned to the devices 1, 2, and 3, respectively, as shown in Fig. 2(c). A hierarchy partition is balanced if the ratio of processing times on the most and least loaded devices is minimized. Although the devices spend similar amounts of time running DNNs, the devices running the DNNs that are used more often will have larger workloads. To accurately account for workloads we must consider the hierarchy structure; i.e. the rate at which DNNs are used. By scaling the DNN processing times by their rate of use, $\langle$root$\rangle$, $\langle DNN_3\rangle$, and $\langle DNN_1, DNN_2, DNN_4, DNN_5\rangle$ are assigned to the devices.

When only workloads are balanced, a single input frame's activation map may be communicated between multiple devices before the output is generated. In the previously de-



DNN	Proc. Time	Rate of Use	Comm. Time
Root	0.038	100%	0.014
DNN_1	0.031	20%	-
DNN_2	0.036	60%	0.020
DNN_3	0.037	20%	-
DNN_4	0.030	20%	-
DNN_5	0.032	20%	-

Fig. 2: (a) The hierarchical DNN structure constructed for the CIFAR-10 dataset using the methods described in Goel et al. [9]; along with (b) the time taken to process a frame (in seconds), the estimated rate of use, and the communication time (in seconds). (c) Hierarchy partition obtained when only balancing DNN processing times. (d) Hierarchy partition obtained when balancing workloads and reducing communication. Dotted lines highlight the partitions.

scribed hierarchy partition, a frame that is processed by the DNNs along the path from the root to DNN_4 is communicated twice. By choosing a partition that minimizes the edge cuts, the communication overhead can be reduced. A balanced minimum cut graph partition is $\langle$root$\rangle$, $\langle DNN_3, DNN_4\rangle$, and $\langle DNN_1, DNN_2, DNN_5\rangle$. This hierarchy partition is depicted in Fig. 2(d). When performing image recognition with this partition, the expected processing time, $\sum$(*DNN processing time*$\times$ *DNN rate of use*), on the three devices are 0.038 seconds, 0.028 seconds, and 0.020 seconds, respectively. Similarly, the expected communication overhead of the hierarchy is given by $\sum$(*Communication time*$\times$*DNN rate of use*) $= 0.018$ seconds.

Having balanced workloads prevents bottlenecks, but may lead to larger communication overheads. Next, we discuss how to find hierarchy partitions that find a tradeoff between the workload balance and communication overhead to maximize the throughput, $T_{th.}$ in eqt. (1).

2) Choosing Hierarchical DNN Partitions: In this section, we use D_i to represent the i^{th} DNN of the hierarchy ($i = 0$ for root), and E_j is a set representing the DNNs assigned to the j^{th} collaborating edge device. $L(D_i)$ is the time taken (latency) by a device to process DNN D_i. The communication time, $C_{i,j}$, is the time required to send an activation map from D_i to D_j, when D_i and D_j are assigned to different devices. $R(D_i)$ is the rate of use of DNN D_i. Because prior studies suggest that the largest hierarchical DNNs have only

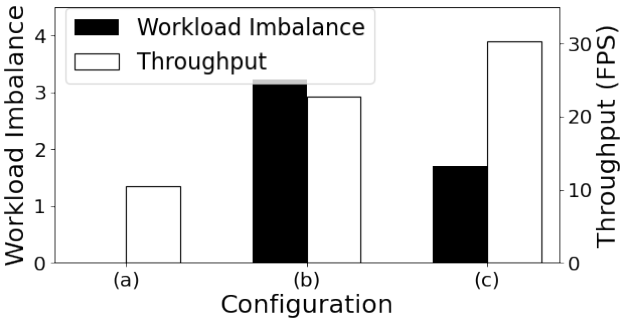


Fig. 3: Impact of hierarchy partitions on throughput and workload imbalance. Workload imbalance is the ratio of the workloads on the most and least loaded devices. Configuration (a) is depicted in Fig. 2 (a) for single-device inference. For three-device inference, configurations (b) and (c) are depicted in Fig. 2 (c) and (d), respectively. Configuration (c) achieves the highest throughput and the most balanced loads.

$\sim$20 DNNs [9] and our experiments consider a maximum of 4 collaborating devices, we use exhaustive search to find the hierarchy partition that maximizes the expected throughput. For larger hierarchies, heuristic algorithms like MeTis [23] can be modified to find partitions without exhaustive search.

For every hierarchy partition, we first calculate the expected processing time on each device E_j as $a_j = \sum L(D_i) \times R(D_i), \forall D_i \in E_j$. The value a_j is the amount of time device E_j takes to process an input scaled by the rate at which the DNNs are used. The largest expected processing time, $\max(a_j)$, is the worst-case DNN processing workload on the devices. Having balanced workloads on devices minimizes $\max(a_j)$. We then obtain the parallel processing overhead of the partition in terms of the expected communication cost as $b = \sum C_{i,j} \times R(D_j), \forall$ DNNs D_i, D_j that have a hierarchy edge that spans devices. Recall that the rate of use of an edge is the same as that of its child DNN.

To evaluate each hierarchy partition, we substitute $\Lambda \times M$ for $\max(a_j)$, the largest expected processing time, and $H \times \tau$ for b, the expected communication cost, in eqt. (1). The estimated throughput is represented in eqt. (2). We select the hierarchy partition that maximizes the throughput, T.

$$T \approx \frac{F}{\Big((F + N - 1) \times \max(a_j)\Big) + \Big(F \times b\Big)} \tag{2}$$

In light of this analysis, we can better understand Fig. 2. Fig. 2 (d) depicts the hierarchy partition obtained by maximizing T for the CIFAR-10 dataset. When $\max(a_j)$ is minimized the workloads are balanced across devices. When b is minimized, the communication overhead is also minimized. Our proposed method finds a tradeoff between the workload balance and communication overhead to maximize the throughput, T. Fig. 3 (c) shows this method processes $\sim$30 FPS with three devices collaborating over Ethernet, thus indicating efficient pipeline parallelism. Fig. 2 (c) shows another hierarchy partition that does not balance workloads or maximize T. The throughput obtained with this partition is 22 FPS, as seen in Fig. 3 (b).

IV. Experimental Results and Analysis

This section experimentally evaluates the proposed method and existing techniques. We vary the hierarchy structures, the edge devices, and the communication medium in our tests. The source code is available on Github [24].

A. Experimental Setup

Platform: We use up to four Raspberry Pi 4Bs in our experiments. The devices communicate using gigabit Ethernet, underneath the ZeroMQ [25] message passing framework. A NETGEAR Nighthawk AC5300 router is used for networking. The hierarchical DNNs open-sourced by Goel et al. [9] are used in our experiments.

Metrics: We evaluate our proposed method based on five metrics: (a) Maximum memory required by a collaborating device for a video frame, measured using the `torchsummary` library; (b) Maximum number of DNN operations (FLOPs) performed by a device for a video frame, measured using the `thop` library; (c) Maximum energy consumed by a device, measured using a Yokogawa WT310E Power Meter; (d) Speed, as latency (inference time for one frame) and throughput (frames per second); (e) Speedup, as the ratio of the throughput obtained with parallel and single-device inference.

Datasets Used: We build and train hierarchical DNNs for three vision datasets: (1) CIFAR-10 [26], (2) SVHN [27], and (3) The random subset of CALTECH-256 [28] used in previous works [2, 9]. The CIFAR-10 and SVHN datasets contain small images (32×32 pixels). Images in CALTECH-256 range from 200×200 to 1024×768 pixels and represent real-life images closely. The Linux `ffmpeg` utility converts images from these datasets into varying-length videos for experiments. For these datasets, the image contents, labels, and hierarchical DNN structures vary significantly [9], allowing us to examine different types of workloads.

B. Experiment 1 - Latency and Throughput

We measure the effect of varying datasets on the proposed approach. Performing pipeline-parallel inference of hierarchical DNNs increases throughput, but the communication overhead may lead to increases in latency. TABLE III shows the impact of the input resolution and hierarchy structure on the latency and throughput of pipeline-parallel hierarchical DNNs. The hierarchical DNN for CALTECH-256 has a maximum depth of 5 and accepts inputs of 224×224 pixels. SVHN and CIFAR-10 have hierarchical DNNs with depth 2 and 3, respectively. With two devices for CIFAR-10, our method has 50 ms latency but achieves 20 FPS because of the inference pipeline. When using three devices, the latency increases to 64 ms and the throughput increases to 30.30 FPS. On one device, the throughput is only 10.55 FPS. Thus, the speedups with two and three devices are 1.90× and 2.87×, respectively. Because the hierarchy for the SVHN dataset has a depth of 2, the speedup is smaller. Although the communication overhead with the CALTECH-256 hierarchy is higher due to the larger resolution, our method achieves a 2.48× speedup with 3 devices. This experiment indicates that the proposed method increases hierarchical DNN throughput for varying hierarchy structures and input resolutions.

C. Experiment 2 - Comparison with Existing Techniques

We compare the proposed approach to the state-of-the-art, summarized in Table I: single-device edge-friendly DNN inference [4], single-device hierarchical DNN inference [9], and parallel DNN inference (data [14, 7], pipeline [8], pipeline+model [13]). The results are tabulated in TABLE IV. Goel et al. [9] is the same as our method for N =1. As the number of devices increases from N = 2 to 4, the memory required on a single device in Zhang et al. [8] remains unchanged. This is because of the large variance in resource requirements across layers; large layers are not split onto multiple devices. Hadidi et al. [13] perform model parallelism to split large layers. However, because of the communication overhead, the time taken does not reduce significantly for $N > 2$. As N increases, the proposed method finds hierarchy partitions that maximize the throughput resulting in significant reductions in processing time. With four devices, MoDNN [7] and MeDNN [14] achieve speedups of 2.03× and 2.43×, respectively. In comparison, our method achieves a speedup of 3.21× with four devices, indicating more efficient parallelism. Results for MoDNN and MeDNN are not reported in TABLE IV because the data is not available. We do not report accuracy because our method does not alter the accuracy of the existing hierarchical DNNs, it only increases efficiency.

TABLE III: Latency, throughput, and speedup obtained with the proposed pipeline-parallel hierarchical DNN method for different datasets. C-256: CALTECH-256.

	Latency (ms)		Throughput (FPS)		Speedup	
	N = 2	N = 3	N = 2	N = 3	N = 2	N = 3
CIFAR-10	50	64	20.00	30.30	1.90×	2.87×
SVHN	43	50	41.32	58.82	1.61×	2.29×
C-256	592	592	2.53	3.99	1.57×	2.48×

TABLE IV: Comparison of FLOPs ($\times 10^6$/frame), memory (MB/frame), energy (J/frame), and throughput (FPS) with different numbers of devices (N) for the CALTECH-256 dataset. Howard et al. [4] is a single-device method so "-" is used for $N > 1$. Blue font indicates the best result.

N	Metric	Zhang et al. [8]	Hadidi et al. [13]	Howard et al. [4]	Our Method
1	#Operations	15.51	4.11	0.58	1.38
	Memory	528.00	98.00	16.00	6.20
	Energy	27.72	14.76	13.86	3.27
	Throughput	0.33	0.35	0.40	1.62
2	#Operations	9.37	2.97	-	0.92
	Memory	521.00	92.00	-	5.00
	Energy	19.72	8.56	-	2.12
	Throughput	0.40	0.58	-	2.53
3	#Operations	6.48	2.39	-	0.55
	Memory	521.00	65.00	-	3.30
	Energy	11.55	8.53	-	1.36
	Throughput	0.49	0.59	-	3.99
4	#Operations	6.48	1.91	-	0.55
	Memory	521.00	65.00	-	2.60
	Energy	10.90	8.46	-	1.04
	Throughput	0.52	0.60	-	5.20

D. Experiment 3 - Evaluation of Theoretical Model

We evaluate the pipeline-parallel hierarchical DNN throughput model presented in eqt. (1). We use Raspberry Pi 3B and 4B boards in this experiment to vary values of Λ (DNN processing times). To vary τ (communication time between devices), we use WiFi communication along with Ethernet. Finally, we also consider different hierarchy structures con-

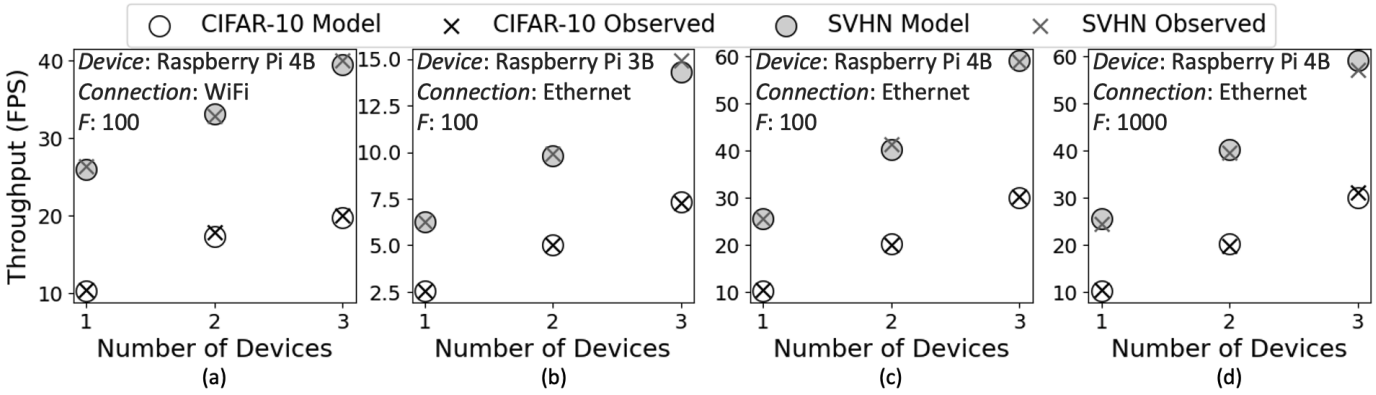


Fig. 4: Evaluation of pipeline-parallel hierarchical DNN throughput model in eqt. (1). (a) Raspberry Pi 4B connected with WiFi. (b) Raspberry Pi 3B connected with Ethernet. Raspberry Pi 4B connected with Ethernet when the number of frames are: (c) $F = 100$ and (d) $F = 1000$. The observed results match the theoretical values closely in different application scenarios.

structed for different datasets. Due to space constraints, we only present results for the CIFAR-10 and SVHN datasets, but we observe similar results with CALTECH-256 as well. For each run, first, a sample input is used to measure the values of Λ and τ. Then, the throughput is measured experimentally and compared with the theoretical value obtained with eqt. (1). Fig. 4 shows that the observed experimental results match closely with the theoretical analysis for four different application scenarios.

V. Conclusions

In this paper, we present a novel method to perform pipeline-parallel inference of a hierarchical DNN for improving the processing throughput on low-power edge devices. Our approach partitions the hierarchical DNN and deploys each partition on a collaborating edge device to allow the processing of multiple frames simultaneously. Existing pipeline-parallel DNN techniques partition conventional DNNs into sets of consecutive layers. These techniques are limited because the large variance in resource requirements and communication costs across layers creates bottlenecks in the pipeline. Through this work, we present a method that partitions hierarchical DNNs to run on multiple devices with balanced loads and decreased communication costs. We first mathematically model the throughput of pipeline-parallel hierarchical DNNs, and then find a hierarchy partition that maximizes the estimated throughput. Our method can find appropriate hierarchy partitions automatically for varying hierarchical DNN structures, edge device specifications, and communication media. Because of the hierarchy partition method, our pipeline-parallel hierarchical DNN achieves significant improvement in throughput with only a small increase in latency. Our experiments confirm that the proposed inference strategy improves the deployability of computer vision on edge device networks, by decreasing the memory, energy, and number of operations on each device.

Acknowledgement

This project was supported in part by NSF CNS-1925713, NSF OAC-2107230, NSF OAC-2104709, and NSF OAC-2107020. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] S. Alyamkin et al. “Low-Power Computer Vision: Status, Challenges, and Opportunities”. In: *2019 IEEE JETCAS*.
[2] P. Panda et al. “FALCON: Feature Driven Selective Classification for Energy-Efficient Image Recognition”. In: *2017 TCAD*.
[3] A. Goel et al. “A Survey of Methods for Low-Power Deep Learning and Computer Vision”. In: *2020 IEEE WF-IoT*.
[4] A Howard et al. “Searching for MobileNetV3”. In: *arXiv:1905.02244*.
[5] S. Aghajanzadeh et al. “Camera Placement Meeting Restrictions of Computer Vision”. In: *2020 IEEE ICIP*.
[6] M. Motamedi et al. “Machine Intelligence on Resource-Constrained IoT Devices: The Case of Thread Granularity Optimization for CNN Inference”. In: *2017 ACM TECS*.
[7] J. Mao et al. “MoDNN: Local distributed mobile computing system for Deep Neural Network”. In: *2017 IEEE DATE*.
[8] J. Zhang et al. “A Locally Distributed Mobile Computing Framework for DNN Based Android Applications”. In: *2018 ACM Internetware*.
[9] A. Goel et al. “Modular Neural Networks for Low-Power Image Classification on Embedded Devices”. In: *2020 ACM TODAES*.
[10] D. Roy et al. “Tree-CNN: A Hierarchical Deep Convolutional Neural Network for Incremental Learning”. In: *Neural Networks* 121 (2020).
[11] A. Goel et al. “Low-Power Object Counting with Hierarchical Neural Networks”. In: *2020 ACM ISLPED*.
[12] J. K. Kim et al. “STRADS: A Distributed Framework for Scheduled Model Parallel Machine Learning”. In: *2016 ACM EuroSys*.
[13] R. Hadidi et al. “Toward Collaborative Inferencing of Deep Neural Networks on Internet-of-Things Devices”. In: *2020 IEEE IoT-J*.
[14] Mao et al. “MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs”. In: *2017 IEEE ICCAD*.
[15] A. Goel et al. “Low-Power Multi-Camera Object Re-Identification using Hierarchical Neural Networks”. In: *2021 ACM ISLPED*.
[16] G. Wang et al. “sensAI: ConvNets Decomposition via Class Parallelism for Fast Inference on Live Data”. In: *2021 MLSys*.
[17] K. Bhardwaj et al. “Memory- and Communication-Aware Model Compression for Distributed Deep Learning Inference on IoT”. In: *2019 ACM TECS*.
[18] J. Geng et al. “Horizontal or Vertical? A Hybrid Approach to Large-Scale Distributed Machine Learning”. In: *2019 ACM ScienceCloud*.
[19] Pasandi et al. “Collaborative Intelligent Cross-Camera Video Analytics at Edge: Opportunities and Challenges”. In: *2019 ACM SenSys*.
[20] Patterson et al. *Computer Architecture: A Quantitative Approach*.
[21] Y. Yang et al. “DEBTS: Delay Energy Balanced Task Scheduling in Homogeneous Fog Networks”. In: *2018 IEEE IoT-J*.
[22] Y. Roh et al. “A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective”. In: *2021 IEEE TKDE*.
[23] G. Karypis et al. *MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0*. 2009.
[24] URL: https://github.com/abhinavgoel95/Pipeline-Parallel-MNN-Tree.
[25] *ZeroMQ*. URL: https://zeromq.org/.
[26] Krizhevsky et al. *Learning Multiple Layers of Features from Tiny Images*. 2009.
[27] Y. Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *2011 NeurIPS*.
[28] G. Griffin et al. *Caltech-256 Object Category Dataset*. http://authors.library.caltech.edu/7694. 2007.