Automated Deep Learning Model Partitioning for Heterogeneous Edge Devices

Arijit Mukherjee
mukherjee.arijit@tcs.com
TCS Research
Kolkata, West Bengal, India

Swarnava Dey
swarnava.dey@tcs.com
TCS Research
Kolkata, West Bengal, India

ABSTRACT

Deep Neural Networks (DNN) have made machine learning accessible to a wide set of practitioners working with field deployment of analytics algorithms over sensor data. Along with it, focus on data privacy, low latency inference, and sustainability has highlighted the need for efficient in-situ analytics close to sensors, at the edge of the network, which is challenging given the constrained nature of the edge platforms, including Common Off-the-Shelf (COTS) AI accelerators. Efficient DNN model partitioning across multiple edge nodes is a well-studied approach, but no definitive characterization exists as to why there is a performance improvement due to DNN model partitioning, and whether the benefits hold for currently used edge hardware & state-of-the-art DNN models. In this paper, we present a detailed study and analyses to address the above-mentioned shortcomings and propose a framework that automatically determines the best partitioning scheme and enhances system efficiency.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

neural networks, deep learning, pruning, nas, edge

ACM Reference Format:
Arijit Mukherjee and Swarnava Dey. 2022. Automated Deep Learning Model Partitioning for Heterogeneous Edge Devices. In *The Second International Conference on AI-ML Systems (AIMLSystems 2022), October 12–15, 2022, Bangalore, India.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3564121.3564796

1 INTRODUCTION

Embedding intelligence at the edge is becoming more critical than ever due to the increasing focus on low latency inference, reliability, data privacy, and sustainability. The practice of sending data packets upstream for analysis within a powerful compute cluster of a data center results in higher latency. Even if the upcoming 5G networks contribute to a lower latency, data carried over cellular networks will always have reliability issues, which may be unacceptable in many real-world scenarios. Further, in certain domains, data privacy is of prime importance, and sending packets over an open network may not be acceptable. In-situ processing close to the data source offers some inherent privacy. All these factors taken together act as a driver for the industry to make the edge computing infrastructure more intelligent.

Across different domains, the most prevalent technique for intelligent data analytics is the use of Deep Neural Networks (DNNs). The need for domain experts for feature extraction and engineering is reduced as a DNN can automatically learn the best features from data. Moreover, the availability of a wide range of pre-trained models has eased the customization of a DNN to a particular task at hand. All of these have accelerated the *time to market* of AI-enabled applications to the wild.

However, the foremost hurdle in this path for fast-track AI deployment is the phase where an AI-enabled application has to be deployed on a resource-limited edge computing setting. The DNN models, often designed by data scientists, in most cases, are not built to cater to resource constraints such as processing power, memory, bandwidth, etc., and hence cannot be directly deployed on constrained edge hardware. The pre-trained large DNNs need to pass through a tedious cycle of pruning, compression, and quantization, while every attempt is made to preserve the accuracy of the models.

One option for running DNN models efficiently on edge hardware is to use COTS hardware AI accelerators which are nowadays available in suitable form factors, *e.g.* USB, PCI, etc. The accelerators, along with the edge hardware, provide a suitable platform for the deployment of AI at the network edge, if certain challenges are addressed. For instance, not all DNN models may fit into an available accelerator of an edge platform and in that case, model developers have to manually partition a model across multiple processing elements, as can be seen in [41]. We have experienced during our experiments that a large ResNext CNN does not fit the Coral TPU unless the host board where it is attached is also used. It is important to understand these *Device Edge* platforms and the challenges in using those before we present our proposed framework.

In this paper, we try to discuss the challenges that are faced in the industrial deployment of intelligent processing at the edge. We also propose a working solution that allows us to dynamically and automatically partition a well-understood pre-trained working model across a set of processing elements at the network edge including COTS accelerators without manual intervention, thereby reducing the time-to-market drastically.

AIMLSystems 2022, October 12–15, 2022, Bangalore, India

ACM ISBN 978-1-4503-9847-3/22/10…$15.00
https://doi.org/10.1145/3564121.3564796

2 CHALLENGES FOR DNN INFERENCING AT EDGE AND A PATH TO SOLUTION

The growing trend for embedding intelligence at the edge, especially in Industrial Internet of Things (IIoT) or Cyber-Physical Systems (CPS) is driven by four major factors [51]: (i) requirement for low latency, (ii) communication cost associated with the transfer of sensed data to the data centers in the cloud, (iii) intermittent packet loss in cellular networks leading to reliability issues, and (iv) privacy issue while transferring the data over the open internet. The drive towards edge computing in the industry has intensified to such an extent that Gartner [54] has predicted that by 2025, around 75% of the data produced by sensors will be processed in-situ at the edge compared to the current level of 10%.

However, in contrast to the controlled environment of data centers and indoor customer premises, such environments present several challenges. Consequently, the properties that are desirable in the *edge grade hardware* are namely:

(1) **Robustness & resilience**, which refers to the capability of an *edge grade hardware* to withstand the difficulties of a real physical deployment environment, such as exposure to shock, vibration, humidity, extreme temperature fluctuations, corrosion, abrasion, etc. Applications that highlight such requirements can be found in various domains, such as remote sensing and earth observation from satellites [15], or robotics [56] etc.
(2) **Wide range of connectivity and I/O options**, with support for several network protocol stacks e.g. WiFi, Bluetooth, optical wireless communication, Zigbee, LoRa, Sigfox, different long-term evolution (LTE) standards, etc., and I/O options to support connectivity to other edge systems and upstream Cloud systems [24, 1, 8]
(3) **Enhanced local processing capability to sustain offline/intermittent Internet connectivity**, for supporting the need for in-place data processing and decision making, apart from collating and sending data upstream [35, 45].

In this work we are interested in the capability of such *edge grade hardware*, characterized above, to efficiently execute AI/ML workloads, especially DNN models. In the recent past several DNN hardware accelerator platforms have been made available for use on the edge. In Table 1 we present a succinct survey of several such accelerators and various applications built on top of those platforms.

The survey categorically finds that these DNN accelerator platforms do not support resilience/ruggedness, connectivity, security, and other important characteristics necessary for real-world field deployment. The current tools and techniques for embedding AI algorithms, specifically DNN models, typically include approaches such as *model reduction* [28, 60, 11, 62], *offloading* the deep learning forward pass to a relatively powerful node in the network edge [52, 33, 48, 61, 38] and *smaller by design* models for low-end devices [37, 13, 25]. However, such tools are often offered in a disjoint manner, and require extensive skills necessary for pruning and compressing a given model in order to fit it into a given platform. This tedious and time-consuming process is further complicated by the absence of an efficient mechanism to map the DNN inference workload to a heterogeneous computing platform, including an embedded host board and bus-connected DNN accelerator(s), to optimize one or more performance metrics, e.g., inference latency, throughput, etc.

2.1 Our Proposition: Automated Fitment with DNN Model Partitioning

To address the challenge outlined above, we propose a method that automatically partitions a DNN model in the best possible manner to fit a heterogeneous computing platform.

Although model partitioning for DNN inference is a well-investigated area of research, we find that there are several shortcomings of the existing state-of-the-art in model partitioning. In this paper, we try to address those from the automation perspective. Our main contributions are as follows:

- It is well known that model partitioning and distributed inference helps in cases where a DNN model does not fit an AI accelerator even after applying model reduction. However, is there any performance gain due to partitioning even if the model fully fits into an AI accelerator? Why would one partition between an AI accelerator and a host embedded board, when the accelerator has a much lower inference latency? Taking these decisions and accordingly partitioning a DNN model graph becomes increasingly difficult when inference is distributed on more than two devices. Such scenarios often occur when DNN inference is partitioned between on-premises networked edge devices. In this paper, we perform this characterization. We demonstrate the benefits of DNN model partitioning using a real-life application scenarios.
- No study has been done so far on the combination of new-age edge platforms and the state-of-the-art DNN models, under partitioned inference, which we do.
- We present a Dynamic Programming (DP) based automated model partitioning algorithm that can automatically partition a DNN model into more than one partition, taking into account the processing capability and connection speed of the heterogeneous processing elements. This algorithm scales well for many processing elements, and its simple implementation makes it suitable for even dynamic partitioning of inference workload on an embedded system.

3 RELATED WORKS

A focused survey on DNN model partitioning reveals several gaps in the current state of the art and acts as a basic step for exploiting the parallelism of a DNN inference workflow.

In [53, 32], the benefits of a layer-wise partitioned inference were first highlighted, for Convolutional Neural Networks (CNN). DNN graph is inherently sequential. However, model partitioning can help in improving several performance parameters through distribution and pipelining [18, 21, 19, 17]. Very recently, authors in [41] used model partitioning to achieve performance improvement of an Unmanned Arial Vehicle (UAV) with a host board (JetSon Nano) and an Intel NCS AI accelerator. However, the partitioning approach presented in the above research requires several manual analyses and interventions, during the design phase.

In contrast, we aim for a fully automated partitioning of the model, given platform constraints and the DNN graph. Towards

Company	DNN Acceleration	Hardware & Associated Research
Intel	[Y] - OpenVino Toolkit	NCS [4, 58, 2], FPGA [59, 10], Nervana
NVIDIA	[Y] - NVIDIA Deep LearningSDK, JetPack SDK	NVIDIA DRIVE [49, 44, 27], Atlan, Jetson
ST Microelectronics	[Y] - Eclipse SPC5Studio.AI plugin	SPC58 Automotive MCU [12, 46]
NXP	[Y] - eIQ Auto Deep Learning - Vision, ML, sensor fusion	S32V2, i.MX RT1060 [6, 47, 5]
Qualcomm	[Y] - Snapdragon Neural Processing SDK	CPU, DSP GPU [42, 3, 40]
Samsung	[Y] - Samsung Neural SDK	Exynos [43, 23], 9810 [22, 14]
Mediatek	[Y] - NeuroPilot AI platform	APU [30, 36]

Table 1: Deep Neural Network Acceleration: Options in Edge Grade Hardware

that, we use a Dynamic Programming (DP) based algorithm. DP approach for DNN partitioning is a well established research area [64, 31, 63, 39]. The main motivation for using this over Reinforcement Learning or Evolutionary algorithm-based partitioning is its simplicity and resource-friendly implementations. This allows a DP algorithm to run even on embedded systems when there is a hardware reconfiguration during runtime, and the inference workload needs to be re-partitioned. In this work, we demonstrate that our DP-based partitioning achieves *pipelined parallelism* and minimizes the make-span of DNN inference tasks on actual embedded SoCs with DNN accelerators, which is not done in any of the earlier works.

4 INFERENCE WORKLOAD DISTRIBUTION

Our goal in this paper is to present a design of a DNN model partitioning algorithm that achieves a balanced workload distribution between embedded SoCs with DNN accelerators for an edge platform to address a major concern of the industry.

We assume that before the partitioning algorithms run, a DNN model is optionally reduced by using any or all of the pruning, compression, and quantization approaches. After generating a set of optimized layer configurations, the partitioning algorithm is applied. Note that partitioning does not change the operation semantics, and hence there is no accuracy drop due to that.

Once a set of optimal configurations are found for each layer of the DNN model, for each of the processing elements of the heterogeneous computing system, our aim is to partition these sequences of layers into sub-sequences and assign those to individual processing elements, such that the DNN inference workload is balanced. Our target is to achieve a static partitioning of the DNN graph automatically, during its deployment on a heterogeneous compute platform which may include edge devices and accelerators.

The layers in a Deep Learning model are essentially sequential. However, a judicious layer partitioning aided with proper distribution of the subset of layers on multiple devices can have a positive effect on the overall average latency of the inference tasks by minimizing the make-span of each task and also improving the overall resource utilization due to the pipeline effect. For the common vision applications in embedded IoT or CPS domains (*e.g.* autonomous cars [55], fire detection using drones/robots [41] etc.), images from sensors are processed individually in-situ as they arrive, typically in batches of one (since these systems generally receive inputs from one or two sensors) [50, 26]. In such scenarios, minimizing the latency also impacts the overall throughput by increasing it, and hence, for some applications, especially in the embedded domain, we claim to improve the overall throughput as well.

The number and capability of underlying processing elements and the bandwidth of the bridge between host and DNN accelerators determine the optimal partitioning of the model into sub-sequences of layers. For instance, the bandwidth between host processor cores is in the order of gigabits per second, whereas the bandwidth between a host and USB accelerator can be a thousand times less. In Fig. 1, we provide a conceptual explanation of our motivation for exploiting the effects of *pipelined parallelism* by partitioning the model layers and assigning them to different devices using an image identification application. In this example, we assume that the model consists of layers $\{L_1, L_2, \ldots, L_6\}$ which are partitioned into three subsets, namely $\{\{L_1, L_2, L_3\}, \{L_4, L_5\}, L_6\}$. Each subset is assigned to one of the three available devices depending on the device parameters, and the pipelined parallelism leads to an obvious reduction in inference latency.

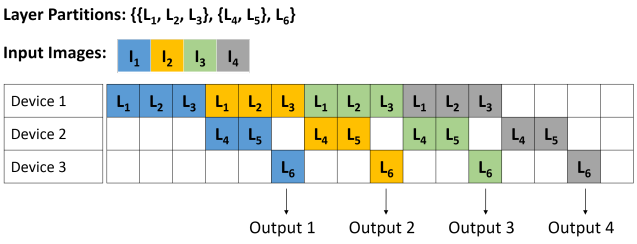


Figure 1: Conceptual depiction of the pipelined parallelism achieved by layer splitting

Using a *brute force* approach, this can be achieved by trying out all possible subsequence combinations on all possible processing elements in a heterogeneous computing platform, resulting in a combinatorial explosion when the number of layers is large. A *capacity based* partitioning approach [20] can be employed by assigning the largest possible sub-sequence of layers to the current processing element, considering one processing element at a time. However, this greedy approach does not necessarily assign a balanced workload (layers) to the host and the accelerator(s). In this work, our goal is to achieve generalized partitioning of a DL model between more than one resource.

Workload distribution and scheduling of DNN execution graphs are necessary to optimize a DNN model for a target platform [57, 7, 9]. Specifically, using the Dynamic Programming (DP) approach for

DNN partitioning is a very well-investigated area [64, 31, 63, 39] which guarantees the optimal assignment in these types of linear partitioning problems [29], if there exist optimal assignments for the sub-problems and the optimization space is relatively smaller.

In [64], a DP based formulation is proposed for latency improvement for both *parallel* and *pipeline* cases. The system is evaluated with virtual machines with server-grade CPU and 64GB RAM, emulating IoT devices. Moreover, this work does not report any latency improvement results in a *pipeline* setting. In both [31] and [63] DP is used for formulating the DNN inference offloading problem for networked Edge devices. In [31] the primary focus is on addressing the problem of deploying a large DNN model on an Edge node by partitioning the model graph as per the capacities of the computing hierarchy. The DP formulation here considers the performance benefit, as well as the uploading overhead of the layers. [63] improves the fused-layer partitioning introduced in [64] and aims for minimizing the total execution time for a single inference. This work reports all the results on a set of virtual machines with homogeneous capacity. In [39] the optimization formulation considers highly resource-constrained devices and includes the memory-related constraints. It shows better performance over several standard Deep Learning frameworks. However, this work also uses a set of identical Linux PCs to prove their system.

None of the earlier works attempt to automate the optimal assignment for DNN layer sequences on an embedded *system on a chip* (SoC) that has attached DNN accelerators of heterogeneous capacities. Our partitioning formulation is based on the solutions to linear partitioning problems [29], and mapping that to DNN layers. However, we are the first ones to demonstrate that such partitioning achieves pipelined parallelism and minimizes the makespan of DNN inference tasks on actual embedded SoCs with DNN accelerators.

The main objective here is to minimize the assignment that takes the most time, including the time required for processing and data transfer, in some processing elements. In this section, we first formulate the problem in a recursive manner and then solve it in an iterative fashion by using a tabular method. The tabular method ensures that smaller problems are solved first and contributes to the overall solution in a bottom-up manner. We assume that the assignment of a sub-sequence of layers to a single processor can be trivial, and can be considered as the base case.

Let us consider a set P of M processing elements (PE) connected over a bus where $p_k \in P$. Let $\mathcal{N}$ be a Deep Learning model graph and $L = \{l_0, l_1, \ldots, l_n\}$ be a set of N layers. Let $T^k_{h,j}$ be the computation latency for processing a sub-sequence of layers h through j in a processing element k and D_j be the data transfer latency for sending the output tensor from layer j to processing element $k+1$. The computation latency is derived for the best configuration selected. Our goal is to automatically create a static partition graph of N layers of the DNN model into M sub-sequences, and assign each partition to a processing element in P as shown in Figure 2.

(1) Base case: When the sub-sequence $(l_1, \ldots, l_j)$ to be assigned to a single processor is given as $O^1_j = T^1_{1,j}$. The tables shown in Figure 2, part (b), depicts the cost matrix used typically for solving using dynamic programming with rows, columns and cells representing layers, processing elements and optimal assignment i.e., partition points of $\boldsymbol{l}$ layers on $\boldsymbol{p}$ processing elements ($\boldsymbol{O^p_l}$), respectively. The base case assignment is equivalent to filling up the first column of this table with the layer latency of $\mathcal{N}$.

(2) Recursive step: The optimal assignment of $(l_1, \ldots, l_j)$ on $(p_1, \ldots, p_k)$ requires the optimal assignment of $(l_1, \ldots, l_{i-1})$ on $(p_1, \ldots, p_{k-1})$ and then the assignment of $(l_i, \ldots, l_j)$ on p_k can be defined as:

$$O^k_j = \min_{i=1,2,\ldots,j-1} \max(O^{k-1}_{i-1}, T^k_{i,j} + D_j) \quad (1)$$

Our dynamic programming-based optimal assignment of layer sequences to processors is depicted in Algorithm 1.

Algorithm 1: Dynamic Programming Based Layer Sequence Assignment to a Set of Processors

Data: List L of execution latencies for each layer in each processor, Output tensor transfer latency D, Image transfer latency I
Result: An assignment array that contain the processor number corresponding to each layer in ascending order

for l = 1,2,. . .,N **do**
  for p = 2,. . .,M **do**
    $O[l][p] \leftarrow 0$;
  end
end
for l = 1,2,. . .,N **do**
  $O[l][1] \leftarrow \sum_{i=i}^{l} L[l][1]$;
end
for p = 1,2,. . .,M **do**
  $O[1][p] \leftarrow L[1][p] + I$;
end
for l = 2,. . .,N **do**
  for p = 2,. . .,M **do**
    $O[l][p] \leftarrow \infty$;
    for i = 1,2,. . .,l-1 **do**
      $\text{cost} \leftarrow \max(O[i][p-1], \sum_{t=i+1}^{l} L[t][p] + D[j])$;
      if $cost \leq O[l][p]$ **then**
        $O[l][p] \leftarrow \text{cost}$;
      end
    end
  end
end
for l = 1, 2,. . .,N **do**
  $schedule[l] \leftarrow \underset{p}{\text{argmin}}\, O[l]$;
end

Dynamic programming problems are often solved using a tabular approach. Towards that, we attempt to create a table similar to Figure 2, part (e) with the optimal assignment costs in a bottom-up fashion. To incrementally fill up this table, we have taken the following approach:

(1) **Step 1:** Get the individual layers and profiling information (layer-wise latency & output size) of all the DNN layers on all the processors.
(2) **Step 2:** Algorithm 1 starts with filling the first column, corresponding to the cumulative execution latencies of all the layers on the first processor, and the first row corresponding to the sum of execution latency and the data transfer latency

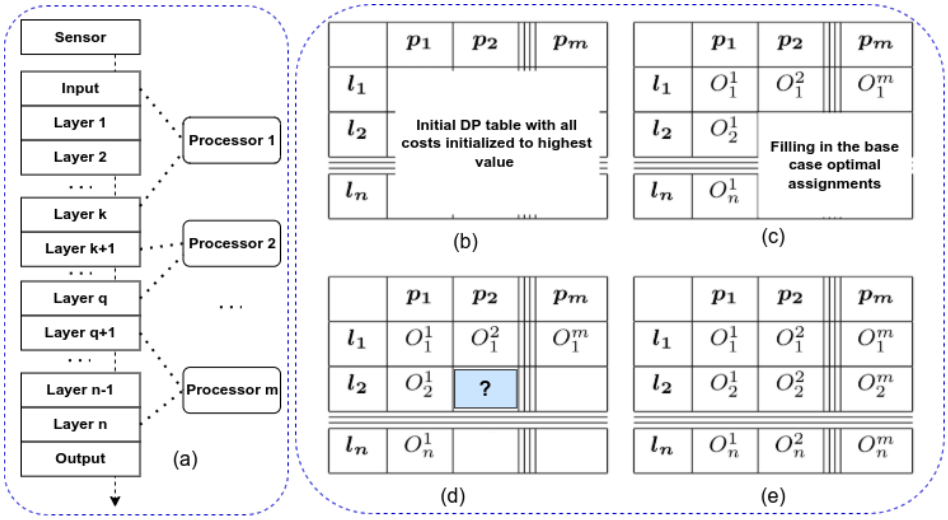


Figure 2: DNN model partitioning for workload balancing: (a) The conceptual view of assignment of layer sequences to Processing Elements, (b) Empty Dynamic Programming table initialized with all costs set to infinity, (c) Filling in the bases cases - column 1 with cumulative execution latencies of all layers on the first processor and row 1 with execution latencies and data transfer latencies of the first layer on all processors, (d) Equation 1 to fill the intermediate costs, and (e) The completely filled static optimal assignment of layer sequences to processors. This table can be used to find layer sequence assignments for each processing element.

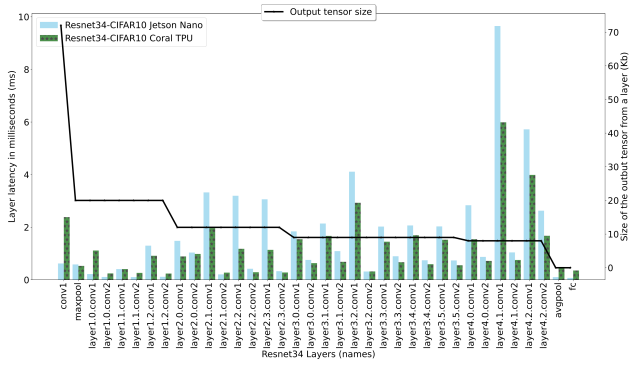


Figure 3: The computation latency of ResNet34 architecture trained on the CIFAR10 dataset, on the two types of processing elements & the output tensor size for each layer. These parameters are the inputs to the Algorithm 1. Similar results are seen for other networks such as ResNext.

of the first layer, on all the participating processors (Fig. 2, part (c)). This is the base case Algorithm 1.

(3) **Step 3:** After that it implements Eqn. 1 to get the schedule for each processor in a recursive, bottom-up manner as shown in (Fig. 2, part (d)), and finally fills up the complete table (Fig. 2, part (e)).

(4) **Step 4:** This table is re-traced to get the optimal schedule of each layer. Specifically, we create a *schedule* array indexed by layers and assign the processor number to the array location corresponding to the index of a layer. An example of such a *schedule* output is as follows: [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1], where the first four layers are assigned to processor 1 and next seven layers are assigned to processor 2.

5 RESULTS AND DISCUSSIONS

Experimental Setup: For all experiments, we use a workstation with an Intel Xeon machine with 32 cores, 32 GB RAM, and two NVIDIA Quadro P1000 GPU cards, as the host system where partition points are determined. We perform all experiments on DL image classification models with CIFAR-10 [34] and ImageNet [16] as input image data.

5.1 Experiments on Partitioning

After these optimal configurations are identified, the inference workload balance Algorithm 1 takes up those configurations, uses the bandwidth information between the host and the accelerator system, and generates optimal scheduling of the subsequences of layers between the available processing elements. We show the optimal computation latency of each layer of an optimally pruned ResNet34 network in Fig. 3.

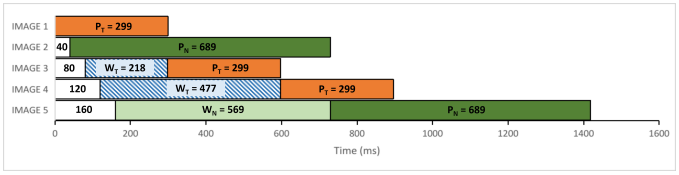
In this figure, we show the layer latencies of both Jetson Nano and the accelerator TPU side-by-side bars. We have marked the output tensor size of each layer as a black line. These parameters are used to find the optimal assignment of layers to devices in line 17 of Algorithm 1. For this model, the optimal schedule is the assignment of the first 16 layers on Jetson Nano and then layers 17 to 35 on the TPU. The optimal execution time (based on the cumulative time of all layers in Fig. 3), `396 ms` is between that of TPU (`299 ms`) and Nano (`689 ms`). This optimal execution time includes the transfer overload of the output tensor after layer 16 and ensures that the

Model	Dataset	Partition Point	Latency Values (ms)		
			Pipeline	JetSon Nano (N)	TPU (T)
ResNet34	CIFAR-10	16	195 (N) + 201 (T)	689	299
ResNet18	ImageNet	7	206 (N) + 207 (T)	636	274
ResNext	ImageNet	10	204 (N) + 204 (T)	588	285

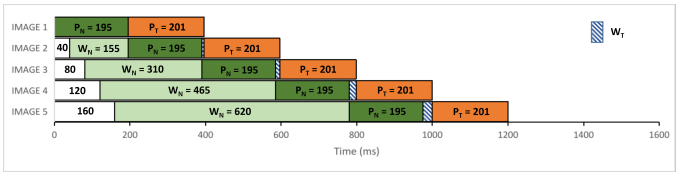
Table 2: Optimal inference scheduling between Jetson Nano & Coral TPU

pipeline efficiency is high during continuous inference through the model.

A sample schedule resembles $[0, 0, 1, 1, 1, 1, \ldots, 1]$, denoting that the first two layers execute on Nano and all subsequent layers are assigned to the TPU. Table 2 presents the detailed results of applying Algorithm 1 on the ResNet34 trained on CIFAR-10 data and two other models trained on ImageNet. The realization of pipelined parallelism by partitioning the model and assignment of subsets of layers onto multiple devices can be seen in the layer execution time graphs in Fig. 4.



(a) Greedy processing on a Jetson Nano and TPU running in parallel



(b) Exploiting pipelined parallelism by executing layer subsets on Jetson Nano and Coral TPU

Figure 4: Comparison of latency between sequential and pipelined inferencing

We experimented on a set of images appearing at various frame rates. In this example, the frame rate is considered to be 25 fps, *i.e.*, each image appears at an interval of 40 ms. Fig. 4a depicts the greedy execution time graph on a Jetson Nano and Coral TPU where each incoming image is either assigned to a processing node is free at that time, or made to wait for an available node. In Fig. 4b, we show the effect of our partitioning approach, where our partitioning algorithm (Algorithm 1) creates a partition point at layer 16 of a ResNet-34 model such that layers 1 to 15 are executed on a Jetson Nano, and layers 16 to 35 are executed on the Coral TPU. It can be seen from Table 2, that the average processing time for the entire model on the Jetson Nano is 688ms, and that on the TPU is 299ms, but our algorithm creates the partition point in a manner such that the average execution time of the layer subset is optimal for each device (195ms to execute layers 1 to 15 on Jetson Nano, and 201ms to execute layers 16-35 on TPU). Hence, in the partitioned execution mode, the waiting time for each image reduces considerably, and an optimal latency is achieved from the fifth image onwards.

Similar experiments were performed by varying the input image rate as well as the number of input images to demonstrate the general applicability of the partitioning algorithm (Algorithm 1) and the plotted latency graph shown in Fig. 5 demonstrates that the pipelined parallelism that we incorporate in the system by judicious partitioning achieves lower latency by executing partition graphs in parallel and thus minimizing the waiting time for the previous processing to complete.

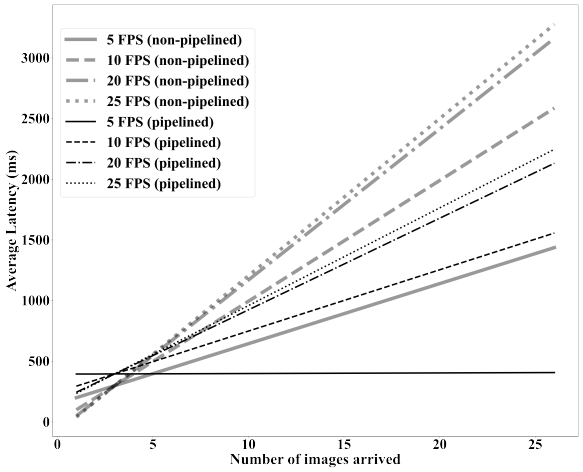


Figure 5: Latency reduction experiment with multiple images arriving at different frame rates

6 CONCLUSION

In this paper, we propose an end-to-end workflow that generates a transformed and accelerated DNN model for all the computing elements, and maps the DNN model layers to the set of computing elements with heterogeneous capacity. We have demonstrated the efficacy of such partitioning and inference workload mapping on an embedded host and a DNN accelerator. We have also presented a short survey, where we have shown that such automation is required for efficiently integrating DNN accelerators with *edge grade* hardware to create typical deployment-ready edge platforms. In the future, we plan to explore and experiment with other classes of DNNs, e.g. Recurrent Neural Networks, Transformer networks, etc., where the DNN graph is not only feed-forward. We believe that this framework, along with DNN accelerator SDKs can together accelerate the deployment of intelligent devices at the network edge.

REFERENCES

[1] Shohin Aheleroff, Xun Xu, Yuqian Lu, Mauricio Aristizabal, Juan Pablo Velásquez, Benjamin Joa, and Yesid Valencia. 2020. IoT-enabled smart appliances under industry 4.0: A case study. *Advanced Engineering Informatics* 43 (2020), 101043.

[2] Ahmed Abdelmoamen Ahmed and Mathias Echi. 2021. Hawk-Eye: An AI-Powered Threat Detector for Intelligent Surveillance Cameras. *IEEE Access* 9 (2021), 63283–63293.

[3] Byung Hoon Ahn, Jinwon Lee, Jamie Menjay Lin, Hsin-Pai Cheng, Jilei Hou, and Hadi Esmaeilzadeh. 2020. Ordering Chaos: Memory-Aware Scheduling of Irregularly Wired Neural Networks for Edge Devices. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. mlsys.org, USA, 44–57. https://proceedings.mlsys.org/paper/2020/file/9bf31c7ff062936a96d3c8bd1f8f2ff3-Paper.pdf

[4] Rosa Andrie Asmara, Bimo Syahputro, Dodit Supriyanto, and Anik Nur Handayani. 2020. Prediction of traffic density using yolo object detection and implemented in raspberry pi 3b+ and intel ncs 2. In *2020 4th International Conference on Vocational Education and Training (ICOVET)*. IEEE, NJ, 391–395.

[5] MANEESH AYI. 2020. *RMNv2: Reduced Mobilenet V2 An Efficient Lightweight Model for Hardware Deployment.* Ph.D. Dissertation. Purdue University. https://hammer.purdue.edu/articles/thesis/RMNv2_Reduced_Mobilenet_V2_An_Efficient_Lightweight_Model_for_Hardware_Deployment/12156771

[6] Maneesh Ayi and Mohamed El-Sharkawy. 2020. Real-time Implementation of RMNv2 Classifier in NXP Bluebox 2.0 and NXP i. MX RT1060. In *2020 IEEE Midwest Industry Conference (MIC)*, Vol. 1. IEEE, NJ, 1–4.

[7] Yixin Bao, Yanghua Peng, Yangrui Chen, and Chuan Wu. 2020. Preemptive All-reduce Scheduling for Expediting Distributed DNN Training. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE, NJ, 626–635.

[8] Maurizio Capra, Riccardo Peloso, Guido Masera, Massimo Ruo Roch, and Maurizio Martina. 2019. Edge computing: A survey on the hardware requirements in the internet of things world. *Future Internet* 11, 4 (2019), 100.

[9] Chunlei Chen, Peng Zhang, Huixiang Zhang, Jiangyan Dai, Yugen Yi, Huihui Zhang, and Yonghui Zhang. 2020. Deep Learning on Computational-Resource-Limited Platforms: A Survey. *Mob. Inf. Syst.* 2020 (2020), 8454327:1–8454327:19.

[10] Mateusz Chmurski, Mariusz Zubert, Kay Bierzynski, and Avik Santra. 2021. Analysis of Edge-Optimized Deep Learning Classifiers for Radar-Based Gesture Recognition. *IEEE Access* 9 (2021), 74406–74421.

[11] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. 2016. Towards the Limit of Network Quantization. *CoRR* abs/1612.01543 (2016). http://arxiv.org/abs/1612.01543

[12] Giulia Crocioni, Giambattista Gruosso, Danilo Pau, Davide Denaro, Luigi Zambrano, and Giuseppe di Giore. 2021. Characterization of Neural Networks Automatically Mapped on Automotive-grade Microcontrollers. *CoRR* abs/2103.00201 (2021). https://arxiv.org/abs/2103.00201

[13] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. 2021. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. In *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica (Eds.), Vol. 3. mlsys.org, USA, 800–811. https://proceedings.mlsys.org/paper/2021/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf

[14] Paul Dempsey. 2018. The teardown Samsung Note9. *Engineering & Technology* 13, 10 (2018), 82–83.

[15] Bradley Denby and Brandon Lucia. 2020. Orbital Edge Computing: Nanosatellite Constellations as a New Class of Computer System. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 939–954.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, USA, 248–255.

[17] Swarnava Dey and Ranjan Dasgupta. 2009. Fast Boot User Experience Using Adaptive Storage Partitioning. In *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. 113–118. https://doi.org/10.1109/ComputationWorld.2009.121

[18] S. Dey, J. Mondal, and A. Mukherjee. 2019. Offloaded Execution of Deep Learning Inference at Edge: Challenges and Insights. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 855–861.

[19] Swarnava Dey and Arijit Mukherjee. 2018. Implementing Deep Learning and Inferencing on Fog and Edge Computing Systems. *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)* (2018), 818–823.

[20] S. Dey and A. Mukherjee. 2018. Implementing Deep Learning and Inferencing on Fog and Edge Computing Systems. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE, NJ, 818–823.

[21] Swarnava Dey, Arijit Mukherjee, Arpan Pal, and P. Balamuralidhar. 2018. Partitioning of CNN Models for Execution on Fog Devices. In *Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing* (Shenzhen, China) *(CitiFog'18)*. ACM, New York, NY, USA, 19–24.

[22] Somdip Dey, Suman Saha, Amit Singh, and Klaus McDonald-Maier. 2020. FruitVeg-CNN: Power-and Memory-Efficient Classification of Fruits & Vegetables Using CNN in Mobile MPSoC. In *2020 IEEE 17th India Council International Conference (INDICON)*. IEEE, NJ, 1–7.

[23] Somdip Dey, Amit Kumar Singh, Xiaohang Wang, and Klaus McDonald-Maier. 2020. User interaction aware reinforcement learning for power and thermal efficiency of CPU-GPU mobile MPSoCs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, NJ, 1728–1733.

[24] Jie Ding, Mahyar Nemati, Chathurika Ranaweera, and Jinho Choi. 2020. IoT Connectivity Technologies and Applications: A Survey. *IEEE Access* 8 (2020), 67646–67673.

[25] Banbury *et al.*. 2021. MLPerf Tiny Benchmark. *CoRR* abs/2106.07597 (2021). arXiv:2106.07597 https://arxiv.org/abs/2106.07597

[26] Dustin Franklin. 2018. NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics. Retrieved March 23, 2022 from https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics

[27] Chang Gao, Antonio Rios-Navarro, Xi Chen, Tobi Delbruck, and Shih-Chii Liu. 2020. Edgedrnn: Enabling low-latency recurrent neural network edge inference. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, NJ, 41–45.

[28] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). ICLR, CA.

[29] Pierre Hansen and Keh-Wei Lih. 1992. Improved Algorithms for Partitioning Problems in Parallel, Pipelined, and Distributed Computing. *IEEE Trans. Comput.* 41, 6 (June 1992), 769–771.

[30] Andrey Ignatov, Cheng-Ming Chiang, and Hsien-Kai et. al. Kuo. 2021. Learned Smartphone ISP on Mobile NPUs with Deep Learning, Mobile AI 2021 Challenge: Report. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, NJ, 2503–2514.

[31] Hyuk-Jin Jeong, Hyeon-Jae Lee, Chang Hyun Shin, and Soo-Mook Moon. 2018. IONN: Incremental Offloading of Neural Network Computations from Mobile Devices to Edge Servers. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 401–411. https://doi.org/10.1145/3267809.3267828

[32] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. ACM, New York, NY, USA, 615–629.

[33] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and S. Mukhopadhyay. 2018. Edge-Host Partitioning of Deep Neural Networks with Feature Space Encoding for Resource-Constrained Internet-of-Things Platforms. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, NJ, 1–6.

[34] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report 0. University of Toronto, Toronto, Ontario.

[35] Yen-Lin Lee, Pei-Kuei Tsung, and Max Wu. 2018. Technology trend of edge AI. In *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, NJ, 1–2.

[36] Chien-Hung Lin, Chih-Chung Cheng, Yi-Min Tsai, Sheng-Je Hung, Yu-Ting Kuo, Perry H Wang, Pei-Kuei Tsung, Jeng-Yun Hsu, Wei-Chih Lai, Chia-Hung Liu, et al. 2020. 7.1 a 3.4-to-13.3 tops/w 3.6 tops dual-core deep-learning accelerator for versatile ai applications in 7nm 5g smartphone soc. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, NJ, 134–136.

[37] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). NeurIPS, Canada.

[38] Jiachen Mao, Zhongda Yang, Wei Wen, Chunpeng Wu, Linghao Song, Kent W. Nixon, Xiang Chen, Hai Li, and Yiran Chen. 2017. MeDNN: A Distributed Mobile System with Enhanced Partition and Deployment for Large-Scale DNNs. In *Proceedings of the 36th International Conference on Computer-Aided Design (ICCAD '17)*. IEEE Press, Irvine, California, 751–756.

[39] Fabíola Martins Campos de Oliveira and Edson Borin. 2019. Partitioning Convolutional Neural Networks to Maximize the Inference Rate on Constrained IoT Devices. *Future Internet* 11, 10 (2019). https://doi.org/10.3390/fi11100209

[40] Mirgahney Mohamed, Gabriele Cesa, Taco S. Cohen, and Max Welling. 2020. A Data and Compute Efficient Design for Limited-Resources Deep Learning. *CoRR* abs/2004.09691 (2020). https://arxiv.org/abs/2004.09691

[41] Arijit Mukherjee, Jayeeta Mondal, and Swarnava Dey. 2022. Accelerated Fire Detection and Localization at Edge. *ACM Trans. Embed. Comput. Syst.* (dec 2022). https://doi.org/10.1145/3510027 Just Accepted.

[42] Qualcomm Developer Network. 2021. Qualcomm Neural Processing SDK for AI. Retrieved July 14, 2021 from https://developer.qualcomm.com/software/qualcomm-neural-processing-sdk

[43] Samsung Newsroom. 2018. Samsung Optimizes Premium Exynos 9 Series 9810 for AI Applications and Richer Multimedia Content. Retrieved July 14, 2021 from https://news.samsung.com/global/samsung-optimizes-premium-exynos-9-series-9810-for-ai-applications-and-richer-multimedia-content

[44] Huy-Hung Nguyen, Duong Nguyen-Ngoc Tran, and Jae Wook Jeon. 2020. Towards Real-Time Vehicle Detection on Edge Devices with Nvidia Jetson TX2. In *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. IEEE, NJ, 1–4.

[45] Arpan Pal, Arijit Mukherjee, and Swarnava Dey. 2016. *Future of Healthcare—Sensor Data-Driven Prognosis*. Springer International Publishing, Cham, 93–109. https://doi.org/10.1007/978-3-319-42141-4_9

[46] Danilo Pau, Marco Lattuada, Francesco Loro, Antonio De Vita, and Gian Domenico Licciardo. 2021. Comparing Industry Frameworks with Deeply Quantized Neural Networks on Microcontrollers. In *2021 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, NJ, 1–6.

[47] SP Kavyashree Prasad and Mohamed El-Sharkawy. 2021. Deployment of Compressed MobileNet V3 on iMX RT 1060. In *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*. IEEE, NJ, 1–4.

[48] Xukan Ran, Haoliang Chen, Zhenming Liu, and Jiasi Chen. 2017. Delivering Deep Learning to Mobile Devices via Offloading. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network* (Los Angeles, CA, USA) *(VR/AR Network '17)*. Association for Computing Machinery, New York, NY, USA, 42–47.

[49] Sergio Márquez Sánchez, Francisco Lecumberri, Vishwani Sati, Ashish Arora, Niloufar Shoeibi, Sara Rodríguez, and Juan M. Corchado Rodríguez. 2020. Edge Computing Driven Smart Personal Protective System Deployed on NVIDIA Jetson and Integrated with ROS. In *Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection*, Fernando De La Prieta, Philippe Mathieu, Jaime Andrés Rincón Arango, Alia El Bolock, Elena Del Val, Jaume Jordán Prunera, João Carneiro, Rubén Fuentes, Fernando Lopes, and Vicente Julian (Eds.). Springer International Publishing, Cham, 385–393.

[50] Jairam Sharma. 2022. Building Industrial embedded deep learning inference pipelines with TensorRT. Retrieved March 23, 2022 from https://learnopencv.com/building-industrial-embedded-deep-learning-inference-pipelines-with-tensorrt/

[51] Duncan Stewart, Jeff Loucks, Mark Casey, and Craig Wigginton. 2019. Bringing AI to the device: Edge AI chips come into their own. https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/ai-chips.html.

[52] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2017. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, NJ, 328–339.

[53] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2017. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 328–339.

[54] Rob van der Meulen. 2018. What Edge Computing Means for Infrastructure and Operations Leaders. https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders.

[55] Wei Wang, Hui Lin, and Junshu Wang. 2020. CNN Based Lane Detection with Instance Segmentation in Edge-Cloud Computing. *J. Cloud Comput.* 9, 1 (may 2020), 10 pages.

[56] Cuebong Wong, Erfu Yang, Xiu-Tian Yan, and Dongbing Gu. 2018. Autonomous robots for harsh environments: a holistic overview of current solutions and ongoing challenges. *Systems Science & Control Engineering* 6, 1 (2018), 213–219.

[57] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, NJ, 392–405.

[58] Y. Xing, P. Kirkland, G. Di Caterina, J. Soraghan, and G. Matich. 2018. Real-Time Embedded Intelligence System: Emotion Recognition on Raspberry Pi with Intel NCS. In *Artificial Neural Networks and Machine Learning – ICANN 2018*, Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis (Eds.). Springer International Publishing, Cham, 801–808.

[59] Mengwei Xu, Yunxin Liu, and Xuanzhe Liu. 2021. A Case for Camera-as-a-Service. *IEEE Pervasive Computing* 20, 2 (2021), 9–17.

[60] Dingcheng Yang, Wenjian Yu, Ao Zhou, Haoyuan Mu, Gary Yao, and Xiaoyi Wang. 2020. DP-Net: Dynamic Programming Guided Deep Neural Network Compression. *CoRR* abs/2003.09615 (2020).

[61] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep Compressive Offloading: Speeding up Neural Network Inference by Trading Edge Computation for Network Latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems* (Virtual Event, Japan) *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA, 476–488.

[62] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (Shenzhen, China) *(SenSys '18)*. Association for Computing Machinery, New York, NY, USA, 278–291.

[63] Li Zhou, Mohammad Hossein Samavatian, Anys Bacha, Saikat Majumdar, and Radu Teodorescu. 2019. Adaptive Parallel Execution of Deep Neural Networks on Heterogeneous Edge Devices. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (Arlington, Virginia) *(SEC '19)*. Association for Computing Machinery, New York, NY, USA, 195–208. https://doi.org/10.1145/3318216.3363312

[64] Li Zhou, Hao Wen, Radu Teodorescu, and David H. C. Du. 2019. Distributing Deep Neural Networks with Containerized Partitions at the Edge. In *2nd USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2019, Renton, WA, USA, July 9, 2019*, Irfan Ahmad and Swaminathan Sundararaman (Eds.). USENIX Association. https://www.usenix.org/conference/hotedge19/presentation/zhou