2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) | 978-1-6654-0424-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/PERCOMWORKSHOPS51409.2021.9430952

MEMA: Fast Inference of Multiple Deep Models

Jeroen Galjaard*, Bart Cox*, Amirmasoud Ghiassi*, Lydia Y. Chen*, Robert Birke†
*TU Delft, Delft, Netherlands. Email: {j.m.galjaard-1,b.a.cox}@student.tudelft.nl {s.ghiassi,y.chen-10}@tudelft.nl.
†ABB Research Switzerland, Baden-Dättwil, Switzerland. Email: robert.birke@ch.abb.com.

***Abstract*—The execution of deep neural network (DNN) inference jobs on edge devices has become increasingly popular. Multiple of such inference models can concurrently analyse the on-device data, e.g. images, to extract valuable insights. Prior art focuses on low-power accelerators, compressed neural network architectures, and specialized frameworks to reduce execution time of single inference jobs on edge devices which are resource constrained. However, it is little known how different scheduling policies can further improve the runtime performance of multi-inference jobs without additional edge resources. To enable the exploration of scheduling policies, we first develop an execution framework, EDGECAFFE, which splits the DNN inference jobs by loading and execution of each network layer. We empirically characterize the impact of loading and scheduling policies on the execution time of multi-inference jobs and point out their dependency on the available memory space. We propose a novel memory-aware scheduling policy, MEMA, which opportunistically interleaves the executions of different types of DNN layers based on their estimated run-time memory demands. Our evaluation on exhaustive combinations of five networks, data inputs, and memory configurations show that MEMA can alleviate the degradation of execution times of multi-inference (up to $5\times$) under severely constrained memory compared to standard scheduling policies without affecting accuracy.**

***Index Terms*—Edge computing, Scheduling, Constrained memory, Memory aware, Multi-inference, Deep neural networks**

I. INTRODUCTION

Deep Neural Networks (DNNs) have been successfully applied to a wide range of applications, from voice assistants to autonomous driving. For the example of the life logging application [1], images of surroundings are periodically captured to record the daily activities. The information about faces, gender, and salient objects are inferred by executing different types of DNNs, such as FaceNet and GenderNet [2]. To accommodate the need to extract multi-faced information from the same image input, multiple DNNs are increasingly bundled together – so-called **multi-inference jobs**.

Recent technological advancements and advantage of data vicinity enable the shift of DNN inference from the cloud to the edge devices [3] that have limited processing power, memory, and energy. It is no mean feat to execute DNNs while simultaneously respecting resource constraints and guaranteeing the performance, e.g., low inference time. This challenge is further exacerbated when encountering multi-inference jobs that need to run multiple DNNs on the same data inputs.

That state-of-the-art actively addresses the challenge of running DNN inference on the resource constrained edge devices from the perspectives of either being memory aware or multi-inference. Indeed, memory consumption is the main concern for executing complex and large-scale DNN on the edge devices [4], [5]. On the one hand, various memory-reduction strategies for single DNN are explored, such as caching technique [6], [7], pruning DNN layers [8] and model compression [4], [5]. On the other hand, efficient resource multiplex over multiple DNNs also sheds light on accelerating the inference performance on edge devices [1], [9], [10], [11] without sacrificing accuracy [12], [13]. While there is little work that takes advantages of memory-aware strategies [14], there is no work that takes into account the effect of different scheduling algorithms and different batch sizes.

In this paper, we advocate to leverage the layer-wise scheduling policy to improve multi-inference jobs. To such an end, we first develop EDGECAFFE [15], a DNN inference framework for dicing the entire DNN into layers and providing configurable layer loading and scheduling policies. Thanks to EDGECAFFE, we empirically evaluate the impact of layer loading into memory and scheduling policies on multi-inference pipelines. We first show that the common practise of loading the entire network at once into memory and executing multi-inference jobs according to the arrival orders fall short in coping with a high number of DNN inferences in small memory space, e.g., 512 MB. We design a novel memory-aware policy, MEMA, which opportunistically interleaves the loading and execution tasks of each convolutional and fully-connected layers[1] based on the estimated memory demands. We extensively evaluate the inference time under different levels of memory configurations and inference scenarios of combining five state-of-the-art CNNs. Our preliminary results show that MEMA can effectively prevent performance degradation, up to 25% for the larger number of networks on extremely constrained memory.

Our contributions are summarized as following. First, we develop a general and flexible layer-wise execution framework, EDGECAFFE (§II). Secondly, we design a novel memory aware multi-inference scheduler, MEMA (§III), which are shown effective compared to the standard scheduling policies. Third, we quantify the impact of different combinations of loading and scheduling policies (§V) via extensive experimental evaluations and rigorous statistical tests.

II. EDGECAFFE AND LAYERED MODEL EXECUTION

To execute and record the performance of multi-inference jobs, we extend the EDGECAFFE [15], which is an in-house

[1]Here, we specifically consider convolutional neural networks (CNN), whose main layers are convolution and fully-connected layers. EDGECAFFE, however, supports for the extension to any Caffe compatible model.

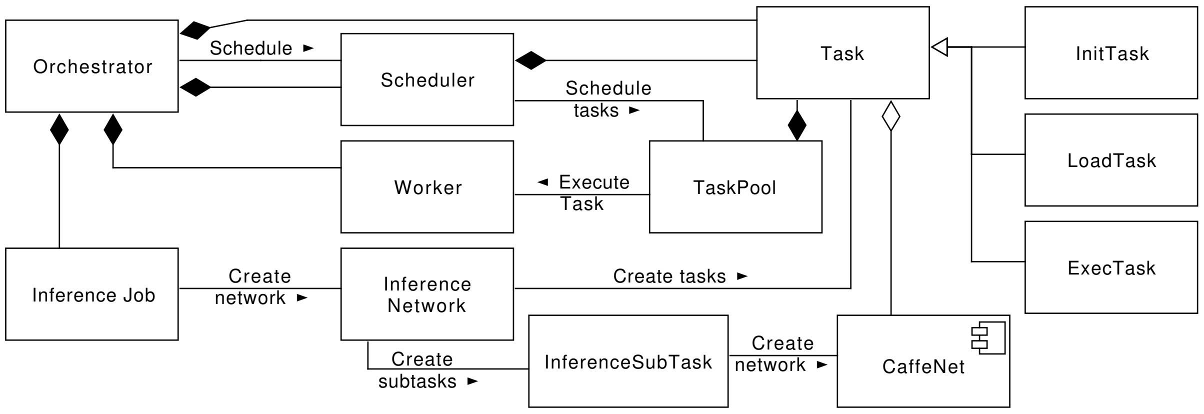

Figure 1: Detailed overview of the EdgeCaffe Orchestrator, Scheduler, and Taskpools. The Orchestrator keeps a collection of InferenceTasks, which are delegated to the scheduler for placement on the available Taskpools. Inference tasks get divided into Network initialization (Init), layer loading (Load), and layer execution (Exec) Tasks.

extension of the Caffe [16] framework which enables layer-by-layer control of DNNs while executing Caffe models. A common practice for executing DNNs is to first load the entire network into memory and process it afterward in one go. EDGECAFFE, instead, provides the flexibility to load and execute networks layer by layer via loading L_i and execution E_i tasks, where the subscript i indicates the i^{th} layer.

Two key decisions to make in EDGECAFFE are (i) when and which task (L or E) to load into memory, and (ii) when and how to schedule the loaded layers. Tasks are created according to a layer loading policy which specifies the execution order of L_i and E_I tasks as dependency graphs (see §II-A). Additionally, a network initialization task I is created to allow for the lazy initialization of the network. Once free, threads in a worker pool select tasks which are ready to run, i.e., having all their dependencies satisfied. We implement different scheduling policies on EDGECAFFE to agilely determine the execution order if multiple tasks are ready to run (see §II-B).

Specifically, the multi-inference jobs are executed by different components of EDGECAFFE shown in Figure 1. First, the inference job is sent to the *orchestrator*, which creates I, E and L tasks based on the requested networks and according to the selected layer loading policy. Once created, the *scheduler* pins the tasks to available *taskpools* and assigns them to worker threads following the selected scheduling policy. The *taskpools* support policies which can schedule independent groups of tasks in parallel, e.g. the DeepEye policy detailed in §II-A.

A. Loading Policies

We implement four loading policies: three policies drawn from state-of-practice and the fourth one providing greater scheduling flexibility.

Bulk. This policy is the status-quo approach used in most frameworks for DNN execution. As shown in Figure 2a, it orders first loading tasks of all layers, before the execution tasks corresponding to the loaded layers become eligible for scheduling.

Linear. Linear loading requires that a layer is only loaded once the preceding layer's output is available. This results in completely interleaved execution of all layers' loading tasks (see Figure 2b).

DeepEye. This policy mimicks DeepEye's execution behaviour [1]. Task dependencies are set such that the fully connected layers can be loaded in parallel to the execution of the convolutional layers (see Figure 2c). This allows for partial amortization of the loading time of fully connected layers (IO-bound) over the execution time of convolutional layers (CPU-bound).

Relaxed. This policy enables the flexible loading of any layer from different types. As such, the scheduler policy can have a higher degree of freedom to execute the readily loaded layers. For example, task E_i and L_{i+1} can be executed simultaneously on different threads, given the execution order specified in the dependency graph.

Under this policy the *orchestrator* creates tasks such that layers can be arbitrarily loaded upon network initialization (see Figure 2d).

B. Standard Scheduling Policies

We implement three well-known scheduling policies from literature, and design a memory-aware policy described in §III.

First Come First Served (FCFS) is a greedy scheduling algorithm that prioritizes tasks with the longest waiting time. This results in executing tasks in order of submission.

Shortest and Longest Job First (SJF, LJF) greedily schedule tasks in increasing and decreasing order of expected execution duration, respectively. SJF's approach minimizes the average waiting time of tasks, as no task has to wait for the longest available task. Whereas LJF provides a guarantee that the longest tasks have minimal waiting time. Both scheduling policies require estimates on task run times. We infer these from profiling runs.

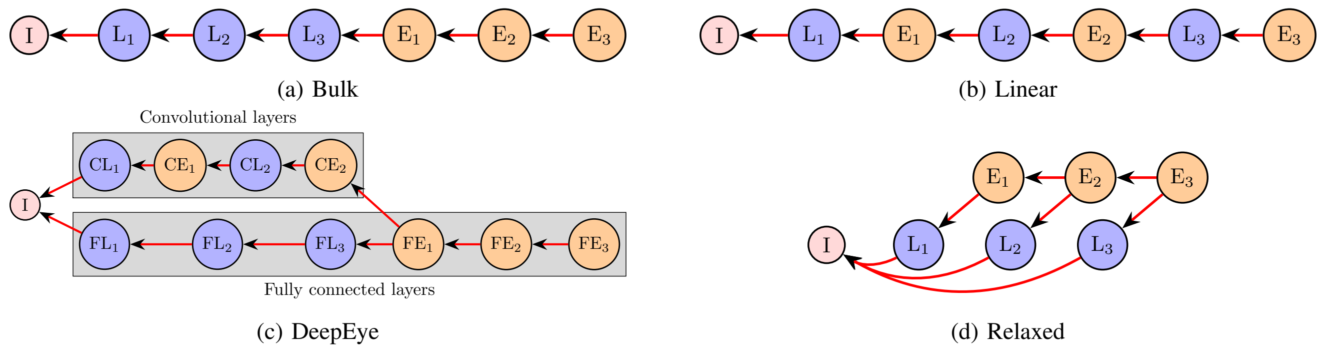

Figure 2: Loading policies ordering Initialization, $L_i$oading and $E_i$xecution tasks for all layers i in a DNN.

III. Memory-aware Scheduler (MeMa)

Ever increasing DNN models challenge the feasibility of running inference on edge devices due to their limited memory space. Multi-inference jobs acerbate this even more. To address this issue, we propose a novel memory-aware scheduling policy, named MeMa. The core design idea of MeMa is to interleave the loading and execution threads of multi-inference jobs while avoiding the memory over-allocation from pre-loading the layers. Algorithm 1 provides the pseudo-code for the MeMa scheduling policy. MeMa does not interfere or alter the structure of the original trained DNN model, hence the model accuracy remains unchanged.

MeMa aims to prevent memory over-allocation and avoid its severe performance penalty during execution. When a process allocates more memory than available, the OS needs to dump currently unused memory pages to disk. When such memory pages are accessed, a page fault is triggered to reload the page from disk. Since disk accesses are orders of magnitude slower than RAM accesses, paging can lead to (severe) execution time penalties. The MeMa policy aims to avoid racking up this penalty by preventing over-allocation, loading only as many layers as fit into the available memory and delaying the others. As memory requirements of even a single layer may exceed the available memory on constrained edge devices, the scheduler is always allowed to load at least one layer in advance.

MeMa tracks the per-layer loading and execution tasks that may be scheduled via two prioritized task lists. The order in which layers are loaded matters. With stringent memory availability in mind, MeMa loads layers in order of appearance in the network, i.e. layer k before layer $k+1$. Hence the loading task list is ordered by layer index. Instead, the ordering of the execution task list is based on expected throughput. This throughput priority is calculated by the expected memory requirement of a layer, divided by the expected computation time of the layer. Since completed execution tasks free up the memory used by the corresponding layer, the throughput metric allows to privilege execution tasks which free up resources faster. Consequently, MeMa can (pre)load layers at a faster pace and increase the chance of "hiding" loading times (typically IO-bound) behind execution times (typically CPU-bound). We infer the layer memory usage and execution times from profiling runs.

Algorithm 1 Pseudo code of MeMa scheduling policy.

```
1:  procedure InitInferenceTask
2:      $inferenceTasks \leftarrow$ NextJob()
3:      Insert $task.loadTasks$ into $load$ priority queue
4:      Insert $task.execTasks$ into $exec$ priority queue
5:  end procedure
6:  function ReadyTasks
7:      while CanStartNewInference() do
8:          InitInferenceTask()
9:      end while
10:     $execTasks \leftarrow$ ReadyExecuteTasks($exec$)
11:     $loadTasks \leftarrow$ ReadyTasks($load$)
12: return $loadTasks, execTasks$
13: end function
14: procedure MeMa
15:     $exec, load \leftarrow$ ReadyTasks()
16:     Add $exec$ to FCFS worker queue
17:     while !OverspendMemory() do
18:         Add NextTask($load$) to FCFS worker queue
19:     end while
20: end procedure
```

IV. Testbed Setup

Scenarios. We consider two scenarios where DNN inference runs on an image of size 500x500 from the EDUB-Seg dataset [17], [18]. The first scenario focuses where only one image is processed by multiple networks, i.e., a single image and multiple DNNs job listed in Table II. In the second scenario, a job consists of multiple images[2], each of which needs to run inferences on DNNs. Both scenarios evaluate the inference performance in terms of average execution time per job across the combinations of the four loading policies and four scheduling policies from §II and §III, and four memory sizes. In addition, for the scenario of multi-image multi-inference jobs, we consider running one or two networks in parallel as well as four batches sizes of images, namely 1,

[2]we term the number of images as the batch size

Table I: Run-time configurations.

Parameter	**Values**
Memory	1 GB, 512 MB, 256 MB
Scheduling Pol.	FCFS, SJF, LJF, MEMA
Loading Pol.	Bulk, Linear, DeepEye, Relaxed
Parallelism	4 (concurrent networks)
Batch size	1, 2, 4, 8 (images)

Table II: Overview of used DNNs; conv.: convolutional, fc: fully connected.

Model	**Size (Disk)**	**Architecture**
AgeNet [2]	45.6 MB	conv:12 fc:8
GenderNet [2]	45.6 MB	conv:12 fc:8
FaceNet [21], [22]	227.5 MB	conv:16 fc:8
SoS [23]	227.5 MB	conv:16 fc:6
SoS_GoogleNet [23]	23.9 MB	conv:10 fc:142

2, 4, and 8 images per job. Table I summarizes all the test variables. Each experiment was repeated 20 times.

DNNs. We consider five DNN models, namely AgeNet, GenderNet, FaceNet, SoS and SoS_GoogleNet, for both scenarios. These models were chosen as they can be deployed for realistic applications, e.g. life-logging [19]. The objectives and architecture of those networks are summarized in Table II.

AgeNet and GenderNet [2] estimate the age and gender of subjects in images. We used the implementation provided by the original authors available on [20].

FaceNet [21] specializes in detecting faces in different poses in images. We used the implementation by Guo [22].

SoS and SoS_GoogleNet [23] perform salient object subitizing. The networks are based on the AlexNet [24] and GoogleNet [25] structures, respectively. We used the implementation from [26].

System specifications. Experiments are performed on Raspberry Pi's, which ran ran Ubuntu 18.04 with 4 cores, and 2 GB and 4 GB RAM. The implementation of EDGECAFFE is based on Caffe V1.0.0 with OpenBlas v0.2.20 as backend. In addition we use Linux cgroups to limit varying levels of available memory and set the number of worker threads to four.

V. EVALUATION

This section first present the impact of limited memory on multi-inference jobs for the scenarios of single image before moving to scenarios of multiple images. We aim to show how different combinations of loading and scheduling policy, including the proposed MEMA, can combat the limited memory space.

A. Single Image Multi-inference Jobs

We start by investigating the effects of constraining memory on single-image jobs, i.e. executing 5 DNNs inference on one image only. For each neural network in Table II, we run inferences on all combinations of loading and scheduling policies and measure the average execution time. To highlight the impact of the different loading/scheduling policies under diminishing available memory, we normalize each result with respect to bulk loading and FCFS scheduling for each memory size. Such a configuration represents the state-of-practice of most deep learning frameworks [1]. Table III presents the results averaged across all five neural networks on Raspberry Pi.

Table III: Normalized execution time relative to bulk loading with FCFS scheduling (status-quo) for each memory size. Lower the values, lower the average execution times.

Loading	**Scheduling**	**2GB**	**1GB**	**512MB**	**256MB**
Bulk	FCFS	**1.00**	1.00	1.00	1.00
	LJF	1.06	**0.94**	1.02	1.05
	MEMA	1.02	1.21	**0.45**	**0.61**
	SJF	1.04	0.96	0.97	1.04
Deepeye	FCFS	**1.10**	1.06	0.62	0.79
	LJF	1.14	1.03	0.62	0.81
	MEMA	1.14	1.11	**0.44**	**0.56**
	SJF	1.15	**1.02**	0.63	0.80
Linear	FCFS	**1.12**	**1.06**	**0.39**	**0.57**
	LJF	1.14	**1.06**	0.40	**0.57**
	MEMA	1.17	1.11	**0.39**	0.59
	SJF	1.13	1.07	**0.39**	**0.57**
Relaxed	FCFS	1.20	**0.99**	0.99	1.04
	LJF	**1.19**	1.04	1.01	0.99
	MEMA	1.21	1.09	**0.44**	**0.57**
	SJF	1.26	1.09	1.18	1.01

Loading policies, such as bulk and linear, enforce strict execution orders and give little to no freedom on scheduling layers. Consequently, when memory is unconstrained relative to inference on DNNs, i.e. 2 GB and 1 GB, execution times across scheduling and loading policies do not differ as shown by a normalized execution time of ≈ 1.0. When memory is scarce, i.e. 256 MB and 512 MB, linear loading is better because it only requires loading one layer into the memory at a time, which reduces the memory demand. As a result, we see $\approx 60\%$, and 40% improvementt across all scheduling policies for 512 MB and 256 MB, respectively.

The DeepEye loading policy divides the tasks into two groups: one CPU-intensive and one IO-intensive which correspond to convolutional layers and fully connected layers. These two groups can be run in parallel to take advantage of their orthogonal resource requirements and optimize execution time. This speedup is shown in Table III.

Relaxed gives the highest freedom to the scheduler. Consequently, under this loading policy, we observe MEMA outperforms significantly across scheduling policies when memory is scarce, i.e. 512 MB and 256 MB. With sufficient memory, i.e. 2 GB and 1 GB, the differences among scheduling policies are limited.

Moreover, we run four-way analysis of variance (ANOVA) to rigorously quantify the significance and importance of different factors on the DNN inference time, namely memory sizes, loading policies, scheduling policies, and the concurrency levels of networks. Due to the space limit, we omit the presentation of the ANOVA table and only provide the

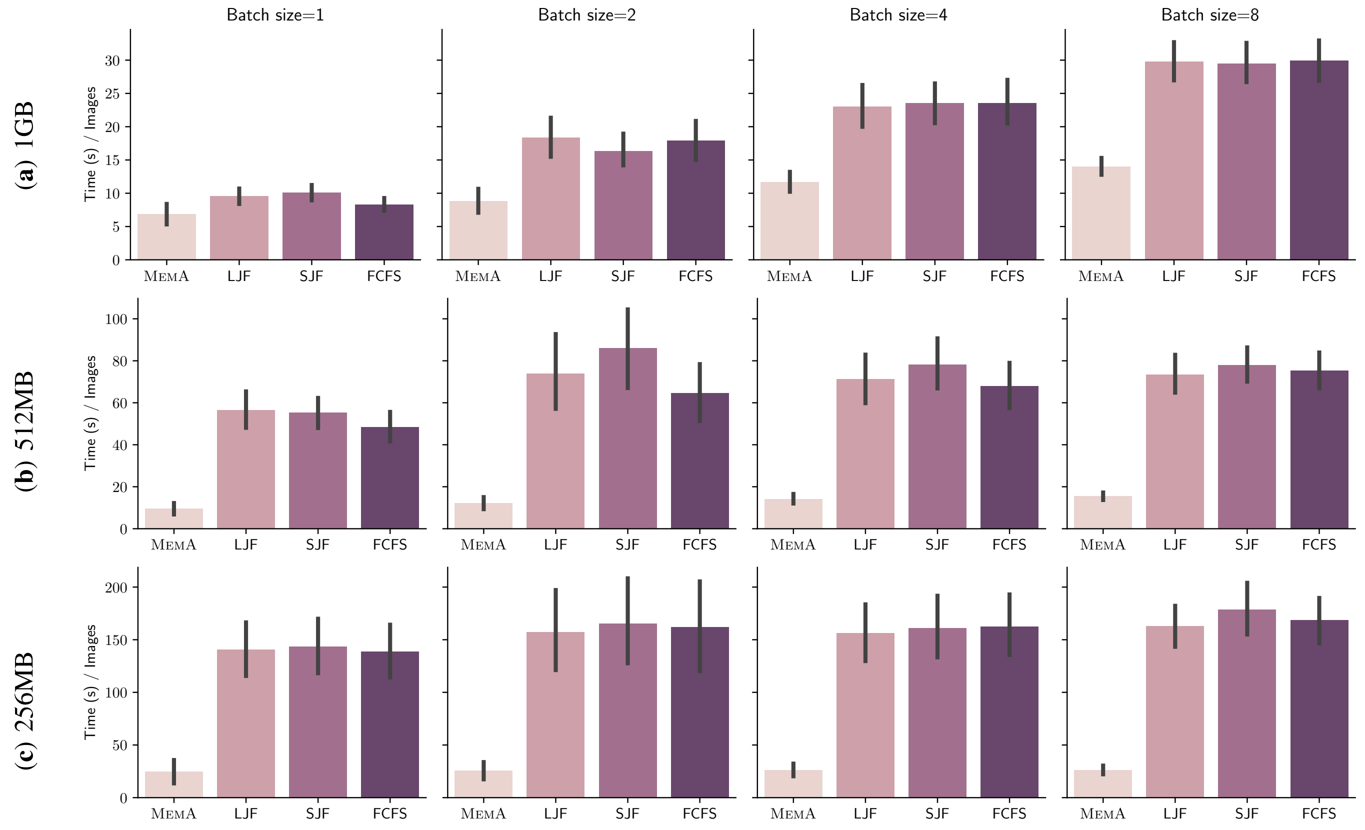

Figure 3: Multi-inference execution time (seconds) under diminishing memory and relaxed loading policy. Each bar represents average execution time of single image (normalized by the batch sizes).

description here. Indeed, all four factors are significant, their resulting F statistics are greater than the significance values of 0.05. In terms of the order of their importance, memory size is the most important one followed by the loading policies, and the scheduling policy and network concurrency are the least important ones. This resonances well with the empirical observations above.

B. Multiple Image Multi-Inference Jobs

Here we evaluate multi-inference jobs which a batch of multiple images are inferred by all five neural networks at a time. Specifically, we consider the batch sizes of 1, 2, 4, and 8 images. We also allow up to two concurrent network inferences to take advantage of available memory, especially in the case of 1 GB. We refer to execution time per job as the time to complete processing a whole batch of images. Due to space constraints we focus on the results with relaxed loading policy, since this policy allows for the highest impact from scheduling, as seen in section §V-A. We consider the same scheduling policies and memory limits as before. Figure 3 summarizes the normalize results, where each bar represents the multi-inference execution time per image. In other words, we normalize the average execution time by the batch sizes.

The impact of scheduling policies increases in the batch sizes and decreases in memory sizes. As shown in Figure 3a, the highest impact of scheduling policies when the batch size is 8 images under 1 GB memory. In term of the absolute time, the difference between MEMA and the second best method is $\approx$ 16 seconds. When memory is restricted further, i.e., 512 MB, memory clearly starts to be scarce. Results in Figure3b show that MEMA performs as the best scheduling policy with the notable difference compared to other methods. As the number of images in each batch increases eightfold, the execution time of MEMA increases by approximately 4 seconds, while other methods increase by an average of 23 seconds under 512 MB memory size. Other scheduling policies have higher execution time under concurrent execution: on average 82% longer executions, i.e, $5\times$ improvement. In addition, the impact of growing batch size when it increases from single image to a larger number is greater than when the batch size is more than one image.

In the most memory constraint case. i.e., 256 MB, the execution time for all scheduling policies almost double due to halving the memory size. Same as the observation for 512MB memory, the impact of increasing the batch size on execution time from 2 images per batch to more is minor compared to increasing from a single image to more than one. This is shown in Figure 3c. Different from LJF, SJF, and FCFS, MEMA keeps the execution times almost constant relative to the batch sizes.

Specifically, in terms of the time difference between batch size 1 and 8, MEMA results into a higher execution time by only 6% on average, while the execution time increases for LJF, SJF and FCFS by 13% , 19% and 17%, respectively. Furthermore, at 256 MB MEMA requires ≈ 133 seconds less on average to complete the multi-inference jobs compared to the second best performing scheduling algorithms. Moreover, MEMA avoids any penalty stemming from concurrent model execution.

As we mentioned, another trend is the impact of increasing batch sizes. Under LJF, SJF and FCFS, one can clearly see that the execution times increase in batch sizes, whereas MEMA is able to keep the increment much lower. Such a contrasting performance is even more visible in cases of stringent memory, i.e., 512 and 256 MB. Additionally, the observed trend for MEMA in Figure 3 is almost constant under 256, 512 MB, while in the case of 1 GB, the trend follows a linear pattern. Compared to all scheduling policies for all the cases, MEMA deteriorates less than LJF, SJF, or FCFS.

Moreover, we validate rigorously our results using three-way ANOVA using memory availability, scheduling mode, and concurrency as predictors for execution time normalized by the batch size. The resulting F-statistics validate the aforementioned observations.

Existing solutions for multi-inference jobs such as NeuOs [12] optimize for throughput and energy usage but do not consider memory. The focus on memory usage within MEMA is beneficial for small devices with limited memory.

VI. CONCLUSION

Motivated by the emerging trend of using multiple neural networks at edge devices, we address the research questions how (novel) scheduling policies can improve the inference performance on memory constrained devices. To such an end, we first develop the execution framework, EDGECAFFE, which enables per-layer execution of DNNs via configurable loading and scheduling policies. To overcome the performance degradation due to the memory constraints, we then design MEMA which aims to effectively schedule multi-inference jobs by opportunistically interleaving the loading and executions of convolutional and fully-connected layers of DNNs according to the estimated memory demands. Extensive evaluation on different combinations of inference networks on Raspberry Pi shows that MEMA can greatly alleviate performance degradation up to 5×, especially in the challenging scenarios of five networks inferring a large number of images under limited memory. For the future work, we will improve the memory estimations of MEMA and explore different edge devices.

VII. ACKNOWLEDGEMENTS

This work has been partly funded by the Swiss National Science Foundation NRP75 project 407540_167266.

REFERENCES

[1] A. Mathurz, N. D. Lanezy, S. Bhattacharyaz, A. Boranz, C. Forlivesiz, and F. Kawsarz, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," *MobiSys*, pp. 68–81, 2017.

[2] G. Levi and T. Hassncer, "Age and gender classification using convolutional neural networks," in *(CVPRW)*, p. 34–42, IEEE, 2015.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ASPLOS*, p. 615–629, ACM, 2017.

[4] W. Yang, L. Jin, S. Wang, Z. Cu, X. Chen, and L. Chen, "Thinning of convolutional neural network with mixed pruning," *IET Image Processing*, vol. 13, no. 5, pp. 779–784, 2019.

[5] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Communications*, vol. 9, no. 1, 2018.

[6] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *MobiSys*, pp. 82–95, 2017.

[7] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: Principled cache for mobile deep vision," in *MobiCom*, pp. 129–144, 2018.

[8] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *ASPLOS*, pp. 907–922, 2020.

[9] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *ACM TECS*, vol. 19, no. 1, pp. 1–28, 2020.

[10] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," *MOBICOM*, pp. 115–127, 2018.

[11] M. LeMay, S. Li, and T. Guo, "Perseus: Characterizing performance and cost of multi-tenant serving for cnn models," in *IEEE IC2E*, pp. 66–72, 2020.

[12] S. Bateni and C. Liu, "Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems," in *USENIX ATC 20*, pp. 371–385, 2020.

[13] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," *MobiSys*, pp. 123–136, 2016.

[14] B. Cox, J. Galjaard, A. Ghiassi, L. Y. Chen, and R. Birke, "Masa: Responsive multi-dnn inference on the edge," in *IEEE PerCom*, p. to appear, 2021.

[15] B. Cox, "Edgecaffe." https://github.com/bacox/edgecaffe, 2020.

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM International Conference on Multimedia*, pp. 675–678, 2014.

[17] M. Dimiccoli, M. Bolaños, E. Talavera, M. Aghaei, S. G. Nikolov, and P. Radeva, "Sr-clustering: Semantic regularized clustering for egocentric photo streams segmentation," *Computer Vision and Image Understanding*, vol. 155, pp. 55–69, 2017.

[18] E. Talavera, M. Dimiccoli, M. Bolanos, M. Aghaei, and P. Radeva, "R-clustering for egocentric video segmentation," in *Iberian Conference on Pattern Recognition and Image Analysis*, pp. 327–336, Springer, 2015.

[19] A. Mathurz, N. D. Lanezy, S. Bhattacharyaz, A. Boranz, C. Forlivesiz, and F. Kawsarz, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," pp. 68–81, 2017.

[20] Y. Jia and E. Shelhamer, "Caffe model zoo," *UC Berkeley*, 2015.

[21] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," in *ICMR*, p. 643–650, ACM, 2015.

[22] A. Guo, "Facedetection_cnn," 2015.

[23] J. Zhang, S. Ma, M. Sameki, S. Sclaroff, M. Betke, Z. Lin, X. Shen, B. Price, and R. Mech, "Salient object subitizing," 2016.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, pp. 1097–1105, 2012.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, pp. 1–9, 2015.

[26] J. Zhang, S. Ma, M. Sameki, S. Sclaroff, M. Betke, Z. Lin, X. Shen, B. Price, and R. Měch, "Salient object subitizing," pp. 4045–4054, 2015.