Multi-inference on the Edge: Scheduling Networks with Limited Available Memory

Jeroen Galjaard[1] , **Bart Cox**[1] , **Lydia Chen**[2] , **Amirmasoud Ghiassi**[2]
EEMCS, Delft University of Technology
[1]{j.m.galjaard-1, b.a.cox}@student.tudelft.nl, [2]{y.chen-10, c.ghiassi}@tudelft.nl

Abstract

The execution of multi-inference tasks on low-powered edge devices has become increasingly popular in recent years for adding value to data on-device. The focus of the optimization of such jobs has been on hardware, neural network architectures, and frameworks to reduce execution speed. However, it is yet not known how different scheduling policies affect the execution speed of a multi-inference job. An empirical study has been performed to investigate the effects of scheduling policies on multi-inference. The execution performance information of multi-inference batch jobs under combinations of loading and scheduling policies were determined under varying levels of constrained memory. These results were obtained using EdgeCaffe: a framework developed to execute Caffe networks on edge oriented devices. Our research showed that a novel scheduling policy, MEMA, can significantly reduce execution speed under stringent memory availability. Overall, this study demonstrates that scheduling policies can significantly reduce the execution speed of multi-inference jobs.

Keywords: Edge computing, Scheduling, Constrained memory, Memory aware, Multi-inference, Convolutional neural networks

Introduction

Deep Neural Networks (DNN) are ubiquitously applied. Where this was previously only possible on dedicated machines, recent technological advancements have resulted in a paradigm shift to edge computing [1]. Such edge devices have limited processing power, memory, and energy consumption available. Due to these limiting factors, it may not be feasible to directly execute DNNs on such devices. This problem gets even more prominent when multiple DNNs are needed for a task, so-called multi-inference jobs. Insight in ways of scheduling multi-inference jobs allows the optimization of inference execution time without making changes to existing DNNs. As a result, allowing for more efficient computation on narrow memory availability.

The optimization of edge-based multi-inference jobs is an actively researched topic. One of the earliest results came from DeepEye, a wearable device capable of efficient offline multi-inference [2]. DeepEye combines several optimization techniques, such as DNN layer compression, network optimization, caching, and a novel layer loading and execution policy [2], [3]. These two latter optimizations combine the execution and loading of DNN layers with orthogonal resource requirements [2]. Effectively loading fully connected layers while executing convolutional layers are being processed. Thereby significantly optimizing resource utilization [3]. As a result, DeepEye gains inference speed-ups of 1.7 and 1.88 times over baseline performance for life-logging and video-assistance scenarios respectively [2]. Other approaches such as NestDNN approach the optimization by the introduction of a hierarchical compressing scheme for DNNs to dynamically trade-off between execution speed and inference quality [4]. Resulting in a significant reduction of the energy footprint and execution time of the evaluated multi-inference jobs [4, p. 124]. Other research by Xian and Kim [5] approaches the optimization process by utilizing offline calculated execution orders for a static set of networks on devices with dedicated GPUs. Their solution, called DART, uses a pre-emptive scheduling scheme for prioritized DNN execution with shared resources [5].

However, it is currently not known yet what the advantages and trade-offs are of different scheduling policies for multi-inference workloads with limited memory. This paper contribution is the generation of insight into the effects of scheduling policies on memory-constrained multi-inference pipelines. It does so using an empirical investigation of the effects on the execution performance of scheduling strategies. These scheduling policies are three well-known policies, as well as a novel memory aware policy. The paper is structured as follows. First, a general background of multi-inference and is provided in section 1. Following that is a synopsis of the papers' contribution. The results of the evaluated experiments are provided in section 3. The results and ethical evaluation are provided in sec-

tion 3 and section 4 respectively. Following that is the discussion of the results in section 5. The conclusion, section 6, and future work, section 7, conclude the paper.

1 Background information

Two multi-inference scenarios were considered in which the execution time of different runtime configurations was recorded. Different kinds of workloads were simulated through the use of five state-of-the-art DNNs. The first scenario is concerned with the execution of small batches, where a single image needs to be executed on a subset of images. In the second scenario, a set of images needs to be run on all five networks. These scenarios ran under different levels of memory availability, ranging from relatively unrestricted to stringent. The execution time of each performed batch was kept track for analysis of performance using multi-way ANOVA tests.

1.1 Convolutional neural networks

The used DNNs, provided in subsection 3.1, are convolutional neural networks (CNN). CNNs are types of DNNs that, in addition to fully connected layers, also contain pooling, subsampling, or convolutional layers. These latter types of layers are capable of extracting increasingly complex features from the input of the network. The output of these convolutional layers is then in turn processed by fully connected layers to make classifications. By their construction, fully connected layers require large amounts of parameters compared with convolutional layers. As a result, the execution of fully connected layers is IO-bound [2]. However, convolutional layers require more computations, making that their processing time is execution-bound.

1.2 Execution pipeline

To execute and record the performance of the multi-inference jobs, the EdgeCaffe[1] [6] framework was used. EdgeCaffe is developed as an extension of the Caffe framework [7] to allow for more fine-grained control of the execution of Caffe models. The status-quo execution of DNNs starts by first loading the entire network into memory and processing it afterward. EdgeCaffe, however, allows for the loading and execution in a layer-by-layer fashion. Because of this, for each layer two tasks are created, a load task and execution task. These tasks are created according to a layer loading policy, as described in subsection 1.3. These layer loading policies dictate the order in which DNN layers may be loaded and executed. Additionally, a network initialization task is created, to allow for the lazy initialization of network input. A more detailed overview the internals of the EdgeCaffe runtime is provided in section 2.

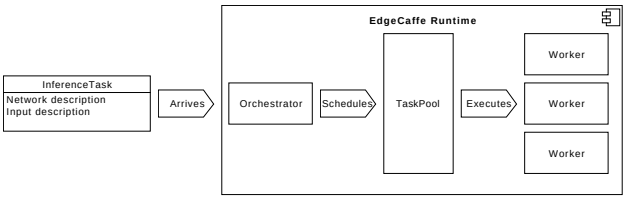


Figure 1: Global overview of EdgeCaffe's runtime structure. Inference Tasks are delivered to the Orchestrator, which controls the placement of sub-tasks in Taskpool, which are in turn consumed by Worker threads.

In the EdgeCaffe runtime, Inference Tasks are managed by an Orchestrator. This Orchestrator places interence job's subtasks in Taskpools for once they are eligible for execution. An abstract representation of EdgeCaffe is provided in Figure 1. To allow for better extension of scheduling policies the EdgeCaffe back-end was extended further to allow for fine-grained control of different scheduling policies. More detail on how the scheduling process is performed is given in section 2.

[1] https://gitlab.com/bacox/edgecaffe

1.3 Layer loading policies

The current state of the art regarding the loading and executing of DNN layers is set apart in this subsection. In EdgCaffe these loading policies dictate dictates in which order tasks can be executed. Allowing for varying degrees of freedom for a scheduling policy to schedule tasks at runtime.

Bulk. The bulk layer loading policy is the status-quo approach in DNN execution. As can be seen in Figure 2, the bulk policy generates task dependencies for an inference job according to two states. In the first stage, the DNN input gets loaded, after which all the network layers must be loaded into memory. Once this first stage is complete, the execution tasks corresponding to the loaded layers become eligible for scheduling.

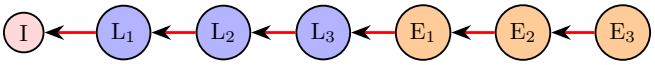


Figure 2: Bulk partitioning layer execution dependency graph. Once the initialization task (I) is executed, the layer loading tasks can be executed (L_x). After completing the L_n tasks, the layer execution tasks (E_y) can scheduled.

Linear. Linear loading imposes task dependencies such that a layer may only be loaded once the output of the preceding layer is available. Loading and executing layers in this fashion results in a completely interleaved execution of the loading and execution of inference sub-tasks. Figure 3 shows a graphical depiction of the linear loading policy.

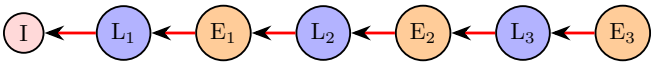


Figure 3: Linear partitioning layer execution dependency graph. Layers are loaded and executed in order of the position in the DNN, requiring the preceding layer to be executed.

DeepEye. When the DeepEye loading policy is enabled, as described in [2, p. 72], the Orchestrator creates task dependencies such that convolutional layer can be executed during the loading of a network's fully connected layers. The DeepEye policy uses one Taskpool to load and execute convolutional layers in a linear order, while a second Taskpool can be used simultaneously to load fully connected layers into memory. Doing so allows for partial amortization of the loading time of fully connected layers over the execution time of convolutional layers [2].

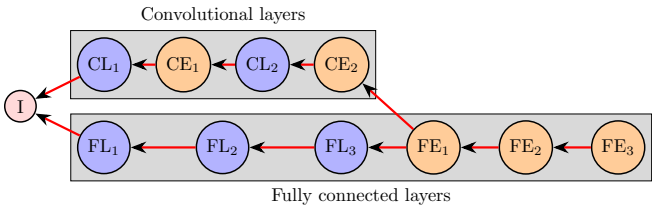


Figure 4: DeepEye layer loading dependency graphs as described by [2, p. 72]. The convolutional layers are loaded (CL_x) and executed (CE_x) in a linear fashion. During the processing of convolutional layers, fully connected layers can be loaded (FL_y) in advance to be executed in bulk fashion later.

2 Scheduling under constrained memory

To allow for different types of scheduling policies to be evaluated, EdgeCaffe was further extended with the SJF, LJF and MEMA policy. To this end a Scheduler class was introduced, which extracts the logic related to Taskpool selection and execution order from the Orchestrator and Taskpools. Additionally, expected memory utilization based on the expected memory usage of tasks was added to the Taskpools.

The execution of an inference job in the extended EdgeCaffe framework is then as follows. First, an Inference Task gets registered by the Orchestrator, which creates tasks according to the set layer loading policy. Once these tasks are created, the execution is delegated to the Scheduler. Which, during inference time, pins the tasks to the available Taskpools. A detailed depiction of the EdgeCaffe runtime is provided in Figure 5.

2.1 Partial loading

To allow for more freedom during task scheduling, the 'partial loading' policy was used. With partial layer loading, the Orchestrator creates inference sub-tasks such that layers can be arbitrarily loaded upon network initialization. To prevent layers from being loaded before its network is initialized, an network and input initialization task must be executed. In Figure 6 a graphical depiction of the EdgeCaffe runtime is given.

2.2 Proposed MEMory Aware Scheduler (MEMA)

Memory Aware (MEMA) scheduling was created with the idea of preventing memory over-allocation during execution. When a process allocates more memory than is available results in having to page to disk. These paging operations result in a time penalty that is orders of magnitudes higher than direct memory access. The MEMA policy aims at preventing over-allocation, delaying the initialization of networks and layers to avoid racking up this penalty. As memory requirements of a layer may exceed the available memory, execution with insufficient memory is allowed to a certain extend. This is achieved by always allowing the scheduler to load one layer in advance. As a result, the order in which layers are loaded matters. With stringent memory availability in mind, it was chosen to load layers in order of appearance in the network.

To keep track of the different loading and execution tasks that may be scheduled, two prioritized lists of tasks are kept track of. One for tracking load tasks, based on layer index, and another for execution tasks, based on expected throughput. This throughput priority is

[2]Memory throughput is calculated as the estimated needed memory over the estimated execution time of the Task.

[3]Returns whether a new network may be started (depending on concurrency) the expected memory utilization does not exceed the expected memory availability

Algorithm 1 Pseudo code for EdgeCaffe's memory aware (MEMA) scheduling policy.

```
1:  procedure INITINFERENCETASK
2:      inferenceTask ← NEXTJOB()
3:      for all task ∈ inferenceTasks do
4:          if task INSTANCEOF(LOADTASK) then
5:              Insert task into load ▷ Prioritity queue using task's layerID.
6:          else
7:              Insert task into exec ▷ Priority queue using task's expected memory throughput[2].

8:  procedure READYTASKS
9:      while CANSTARTNEWINFERENCE() do [3]
10:         INITINFERENCETASK()
11:     execTasks ← READYEXECUTETASKS(exec)
12:     loadTasks ← READYTASKS(load)
13:     return loadTasks, execTasks

14: procedure MEMA
15:     exec, load ← READYTASKS()
16:     for all ∀task ∈ exec do
17:         PINTASK(task)
18:     while !OVERSPENDMEMORY() do
19:         task ← NEXTTASK(load)
20:         PINTASK(task)
```

calculated by the expected memory requirement of a layer, divided by the expected computation time of the layer. The scheduler starts by checking whether new load tasks and execution tasks have become available since the last scheduling round. Loading tasks are then prioritized by the corresponding layer index, forcing layers to be loaded in an in-order fashion. Execution tasks are prioritized separately by the expected throughput of the layer. This throughput of a task is calculated by dividing the required memory by the expected execution duration. Algorithm 1 provides pseudo-code for the MEMA scheduling policy.

As MEMA requires performance data for effective scheduling, data files must be provided to the scheduler. These files are generated offline by evaluating the performance of individual networks on a device. These files were generated by recording the average execution time of each DNN multiple times, in our case in 50 fold. Additionally, the configuration files provide the expected memory requirements for loading and executing each layer. These estimates come in two-fold as well, again for the loading and the execution of layer. The file sizes of the partial network layers were used as estimates for loading a network. For the memory estimates for the execution of layers, the memory analysis performed by Bouwer [8] was used for the five networks.

2.3 Non-memory aware schedulers

First Come First Served (FCFS) is a greedy scheduling algorithm that prioritizes tasks that are the longest waiting inline. The scheduler places the earliest submitted task with completed dependencies on the earliest available TaskPool. In batched execution where tasks are available from the start, this results in the execution in order of submission.

Shortest and Longest Job First (SJF, LJF) greedily schedules tasks in respectively increasing and decreasing order of expected

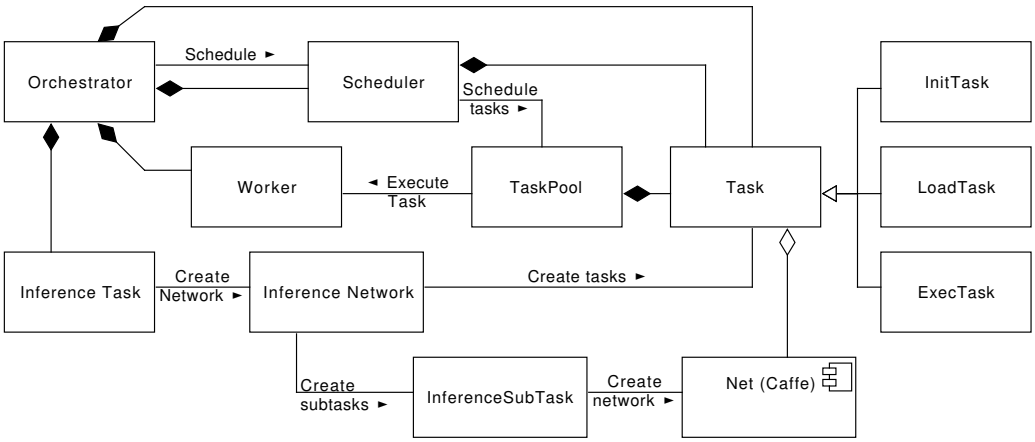


Figure 5: Detailed overview of the EdgeCaffe Orchestrator, Scheduler, and Taskpools. The Orchestrator keeps a collection of InferenceTasks, which are delegated to the scheduler for placement on the available Taskpools. Inference tasks get divided into Network initialization (Init), layer loading (Load), and layer execution (Exec) Tasks.

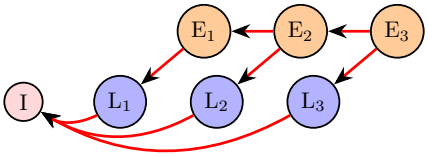


Figure 6: Partial partitioning layer execution dependency graph. After the initialization task (I) is executed, layer loading tasks (L_x) can be scheduled in arbitrary order. Execution tasks (E_y) can be scheduled once the corresponding network layer is loaded.

execution duration. 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 these scheduling policies require additional information for each task in order to schedule effectively.

3 Results

Two types of batched multi-inference were considered. The first scenario evaluated the inference time of running a set of DNNs on an input image (500x500). The second scenario considers the execution of multiple images (500x500) on all five neural networks. Both these types of batched execution were evaluated under varying memory availability, with both serial and parallel execution of networks. A summary of the investigated runtime configurations is given in Table 1.

3.1 Deep Neural Networks

During the evaluation of the scheduling policies, five DNNs were used. The chosen networks are capable of extracting meaning and context from images. An overview of the general description of the architecture of the used networks is provided in Table 2. Here follows a short overview of the used networks.

AgeNet and GenderNet [9]: the AgeNet and GenderNet networks trained to estimate the age and gender of subjects in images. The used implementation was provided by the original authors, which is publicly accessible on the Caffe Model Zoo.

FaceNet [10]: FaceNet is a DNN specialized in the detection of faces in different poses in images. As the implementation by the authors is not available to the public, the implementation by Guo [11] was used.

Salient Object Subitizing [12]: the two used Caffe models for Salient object subitizing were created by the original authors of the accompanying paper. The two networks SoS and SoS_GoogleNet are based on the AlexNet [13] and GoogleNet [14] network structures respectively. The used Caffe models are publicly available through the authors' site.

Parameter	Values/input
Parallelism	1 - 2 (number of allowed concurrent networks)
Memory	1G, 512M, 256M (RAM)
Scheduling	FCFS, SJF, LJF, Memory Aware
Loading	Bulk, Partial, Linear, DeepEye
Small batches	3 - 5 (subsets of the 5 networks)
Large batches	1, 2, 4, 8 (number of images)

Table 1: Runtime configuration parameters of the evaluated small and large batches. Each combination of these values were run in 20 fold to gather data for performance analysis.

Model	Size (Disk)	Architecture
AgeNet	43.5 MiB	$conv^4$:12 fc^5:8
GenderNet	43.5 MiB	conv:12 fc:8
FaceNet	217.0 MiB	conv:16 fc:8
SoS	217.0 MiB	conv:16 fc:6
SoS_GoogleNet	22.8 MiB	conv:10 fc:142

Table 2: Description of the used deep neural networks in the batch execution scenarios. (AgeNet and GenderNet [9], FaceNet [10], [11], SoS and SoS_GoogleNet [12]).

3.2 System specifications

The experiments were performed using virtual machines (VMware), which were executed on HP ZBook G4 Studio equipped with an Intel i7-7700HQ processor and NVME drive. For the small batches two parallel virtual machines were used, each with 2 virtual cores, 4 GB RAM and 40 GB disk space running Ubuntu (18.04). The evaluation of the large batches was run on a single virtual machine, which was provided with 2 additional cores. These machines executed the extended version of EdgeCaffe using Caffe version 1.0.0 with OpenBlas (0.2.20) as backend. To simulate the varying levels of memory availability Linux cgroups were used, which was allowed to use up to 4 GB of memory including swap space.

3.3 Constraining memory

To gain insight into the effects of constraining memory on the individual networks, the runtime of the used neural networks was evaluated under four memory configurations. Table 3 provides an overview of the average speed-up factor on the average runtime a single DNN at a time. The data shows that at relatively unconstrained memory, the partial loading policy is favourable, with the exception of the MEMA policy at 1G. The other layer loading policies also show minor impact by the chosen scheduling policies. Again with the exception of MEMA at 1G of memory.

When considering tighter memory constraints, at 512M and 256M, partial loading paired with MEMA scheduling performs well again. This contrasts with the trend visible in the other scheduling policies with partial loading, which degrade in performance gain quickly. The performance gain of DeepEye stays relatively constant under the various loading schemes.

Layer loading	**Scheduling policy**	**2G**	**1G**	**512M**	**256M**
bulk	fcfs	1.00	1.00	1.00	1.00
	ljf	1.00	1.00	1.00	0.99
	memory	1.01	1.01	1.01	1.00
	sjf	1.01	1.01	1.00	1.00
deepeye	fcfs	0.92	0.91	0.91	0.88
	ljf	0.93	0.92	0.92	0.87
	memory	0.91	1.02	1.00	0.92
	sjf	0.92	0.91	0.91	0.88
linear	fcfs	1.00	1.00	1.00	0.91
	ljf	1.01	1.01	1.01	0.92
	memory	1.01	1.01	1.01	0.92
	sjf	1.01	1.01	1.01	0.92
partial	fcfs	0.77	0.77	0.89	1.15
	ljf	0.78	0.77	0.90	1.16
	memory	0.80	0.92	0.95	0.91
	sjf	0.76	0.75	0.85	1.16

Table 3: Average inference speed gain of runtime combinations of layer loading policies and scheduling policies, relative to Bulk loading with FCFS (status-quo). MEMA performs well at relatively unconstrained as well as stringent memory availabilities. Further, it can be seen that the partial loading scheme combined with greedy scheduling policies deteriorates in its performance.

[4]Convolutional layers (conv).

[5]Fully connected layers (fc).

3.4 Small batch evaluation

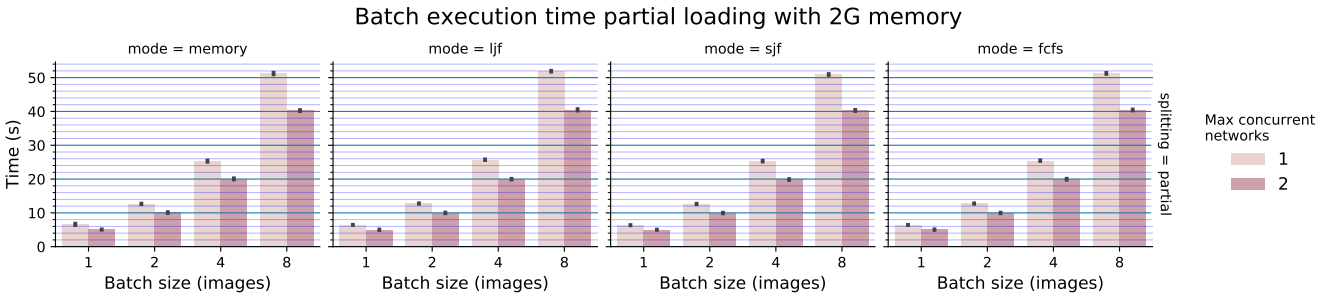
After considering the performance of the DNNs under constrained execution, the small batched jobs were evaluated. As can be seen in Figure 9, it is apparent that decreasing the memory availability negatively impacts the execution runtime. This increase is slight when reducing memory from 2G to 1G. But even more evident when the memory is reduced further to 512M and 256M (see Figure 8). Some even more than doubled in their execution time. At 512M and lower the execution of networks does generally not benefit from executing networks concurrently.

The linear layer loading method shows not to be affected by different scheduling policies. Although its execution time in the unconstrained environment is relatively high, it starts to perform comparably to the other layer loading schemes with limited memory. Bulk, DeepEye, and partial loading show a steeper increase in execution time when memory is restricted when MEMA is not enabled, as can be seen in Figure 9c and Figure 9d illustrate. This trend becomes more evident when parallel execution is allowed, which generally shows a degradation in inference speed. The exception to this is the MEMA policy, which is less affected by allowing parallel execution at 512M (see Figure 9c, and not at all at 256M (Figure 8b).

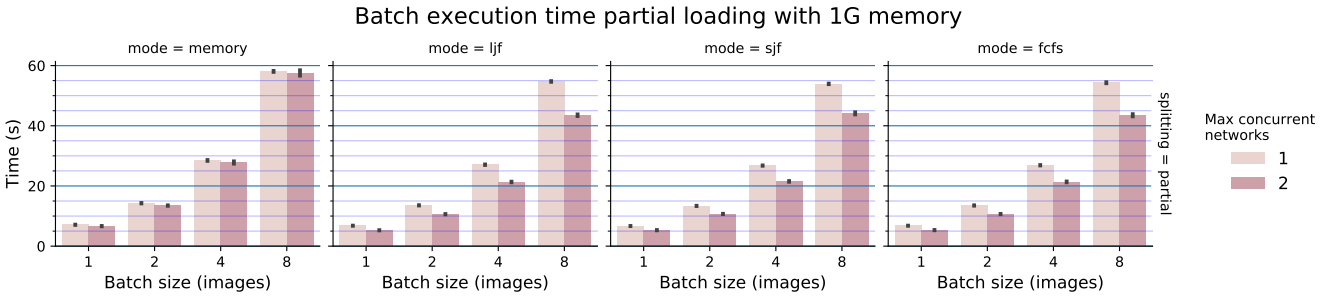
An overview of the performed four-way ANOVA-test is given in Table 4. Its result stresses the effects of limiting memory ($F(2, 30624) = 6581.61$, $p = 0$) as well as the different layer loading policies. Besides, it provides evidence that scheduling policies affect execution time is limited when compared to the contribution that the different layer loading policies provide.

	df	F	PR(>F)
memory	2	6581.61	0.00e+00
splitting	3	89.11	2.06e-57
schedule	3	54.99	1.93e-35
parallel	1	193.49	7.47e-44
memory:splitting	6	123.62	4.69e-155
memory:scheduling	6	44.12	5.08e-54
splitting:scheduling	9	9.76	4.69e-15
memory:parallel	2	205.16	3.11e-89
splitting:parallel	3	23.86	2.05e-15
scheduling:parallel	3	25.88	1.03e-16
memory:splitting:scheduling	18	8.91	8.33e-25
memory:splitting:parallel	6	37.38	1.87e-45
memory:scheduling:parallel	6	20.00	1.83e-23
splitting:scheduling:parallel	9	3.57	1.89e-04
all	18	3.61	3.23e-07
Residual	30624		

Table 4: ANOVA test result of the small batches using the memory available (memory), layer loading policy (splitting), scheduling policy (schedule) and concurrent networks (parallel) as categories for the the execution time. The execution time was normalized by the count of layers within an executed batch. Although significant, the scheduling policy shows a smaller contribution to the explainability than compared to the other runtime parameters.



(a) With 2G of available memory, all scheduling policies perform comparable to another under the provided workloads.



(b) Although the greedy scheduling policies show comparable execution times, the MEMA seems to stagnate at this lower level of memory availability.

Figure 7: Multi-inference execution speed (seconds) under a non memory-constrained execution environment. Each bar represents the average of 20 independent runs. Choosing different scheduling policies at these levels does not show significant improvement over baseline performance. However, MEMA shows an increase in execution time at 1G.

3.5 Large batch evaluation

Batches of 1, 2, 4, and 8 images were run using the partial layer loading policy paired with the scheduling policies. We continue the results by looking into the effects of scheduling policies when larger batches of inference tasks need to be executed under limited memory.

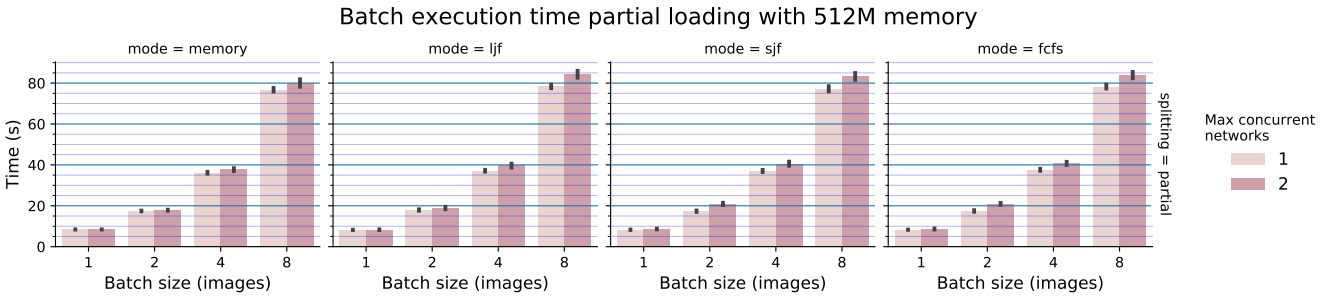
Figure 7 illustrates the limited impact of scheduling policies when the execution is relatively unconstrained. With sufficiently large memory availability, as Figure 7a showcases, all scheduling policies perform similarly. All scheduling policies show a reduction in execution time when networks are allowed to run concurrently. Figure 7b shows a slight increase in average execution time for the batch jobs over execution with 2G (see Figure 7a). However, the MEMA does not show an improvement when compared with the other policies as concurrent network execution is allowed.

When memory is restricted further, as can be seen in Figure 8, the average execution time shows a considerable increase in execution duration. In Figure 8a, all scheduling policies perform comparably and show a global increase in execution time when networks are run concurrently. This penalty of concurrent execution increases as memory is constrained to even lower levels, as can be seen in Figure 8b. Compared to the deterioration in execution speed experienced by FCFS, SJF, and LJF, MEMA experiences this to a lesser degree. At 256M MEMA requires significantly less time to complete inference jobs than the greedy scheduling algorithms.

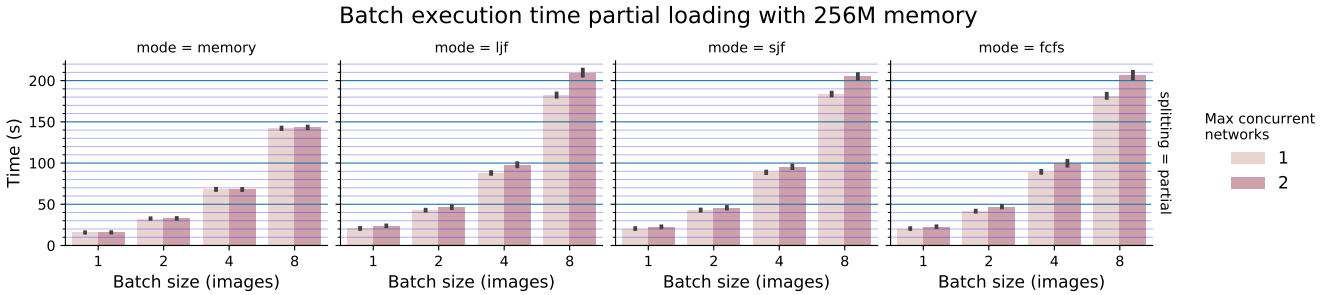
With the collected data a three-way ANOVA test was performed. Using the memory availability, scheduling mode, and concurrency as predictors for execution time normalized by the batch size. An overview of the results from the ANOVA test is given in Table 5. From these results, it can be seen that the trend seen by decreasing

	df	F	PR(>F)
memory	3	46708.28	0.00e+00
scheduling	3	525.91	7.61e-258
parallel	1	35.07	3.69e-09
memory:scheduling	9	603.89	0.00e+00
memory:parallel	3	474.92	4.41e-238
scheduling:parallel	3	14.93	1.26e-09
all	9	34.50	2.15e-57
Residual	2208		

Table 5: ANOVA test summary of the large batches. The execution time was normalized by the number of images in the batch. Using the memory available (memory), layer loading policy (splitting) and scheduling policy (schedule) as predictors for the the execution time. The execution time was normalized by the batch size.



(a) At 512M the networks start to become resource constrained during execution. MEMA scheduling shows a slight edge over the other scheduling policies.



(b) With MEMA scheduling enabled, batches are executed considerably faster. The other greedy scheduling policies perform comparable to one another.

Figure 8: Multi-inference execution speed (seconds) under a resource-constrained execution environment. Each bar represents the average of 20 independent runs. At more constrained memory availability, MEMA shows to significantly decrease execution time.

the memory is confirmed by the ANOVA test. Additionally, it can be seen that the execution time is significantly influenced by the selected scheduling policy under varying levels of memory constraint, confirming the trends observed in Figure 7 and Figure 8.

4 Ethics

During the execution of the research care was taken to localize and address ethical aspects. Given the lack of data collection or human computer interaction during the research, no privacy related issues are relevant. As such we focus on the epistemic problems that relate to the repeatability of the conducted experiments.

The research is based on Caffe the caffe framework and DNNs developed for Caffe named Caffe models. Caffe is open-source and accessible to the public under the 2-Clause BSD License, as such the used Caffe code allows the inclusion in the EdgCaffe framework for research purposes. The used Caffe models needed for workload eveluation were obtained through the the Caffe ModelZoo[6]. The used networks are released under a license that permit re-use and adaption.

The EdgeCaffe framework and source code used for this research is not yet available to the public. Its release to the public is at the time of writing aimed at September 2020, accompanied with a complementary paper evaluating its performance. The EdgeCaffe contains, besides the core of the EdgeCaffe framework, a modified version of Caffe needed for EdgeCaffe bindings. Additionally the scripts used to gather the data are contained there as well. To allow for better reproducibility of the performed work, the repository includes instructions to setup an environment similar to the one used during the evaluation. As such we consider that care has been taken to allow for a repeatable evaluation of memory constrained multi-inference.

[6]https://github.com/BVLC/caffe/wiki/Model-Zoo

5 Discussion

In this paper, the extent to which scheduling policies affect multi-inference execution performance with limited memory was investigated. This was done by collecting the average execution of DNNs on varying degrees of constrained memory. The findings of the study show that scheduling policies can significantly affect the execution speed of constrained memory multi-inference jobs. The effects of scheduling policies become more pronounced when memory is stringent or larger volumes of input are processed. In these cases, conservative layer loading policies such as linear loading, as well as the novel MEMA scheduler have significantly shorter inference speed over baseline performance.

The utilization of the DeepEye policy resulted in considerable performance improvements over bulk execution. However, the speed gain reported by Mathur et al. [2] was not achieved. We consider that this is caused by a lower degree of concurrency, and further evaluation would be needed to confirm this. Other approaches, such as Neurosurgeon [15] and EdgeBatch [16], also focus on energy consumption during execution. Conducting such measurements on execution would be a valuable addition to allow for a more complete evaluation of different scheduling policies. Other resource-aware scheduling policies exist, albeit at different levels of abstraction. MCDNN approaches edge multi-inference with partial cloud offloading by scheduling with the ability to manage power, latency, and monetary budgets for multi-inference tasks [17]. NestDNN attacks it from a different angle, using the dynamic selection of hierarchically compressed networks to schedule within its resource budget [4]. Although these approaches also utilize a resource-aware

scheduling policy, they focus on a network-by-network execution approach. A distributed approach named DeepThings utilizes shared computation to speed up and execute DNNs even exceeding a single device available memory [18]. However, this framework is meant for scheduling tasks on a cluster of Internet of Things (IoT) devices [18].

The performed evaluations and the MEMA scheduler have some limitations. The discovery of execution starvation could not be addressed in the evaluation addressing this issue would improve the performance of the MEMA policy. The performance dip at 1G experienced by MEMA seems to be caused by an underestimation of memory availability. Solutions for these points were drafted up, but could not be included in the performance evaluation due to time constraints. Aside from performance, the scheduler is dependent on offline generated data. Doing so requires a one-time initialization procedure, using a model to estimate layer execution time and memory usage would allow for the more general applicability of the scheduler. The evaluation gives limited insight into benefits from parallelism at higher levels of memory availability. We consider that running evaluations with higher degrees of concurrent execution would be a valuable addition.

6 Conclusion

In this paper, the effects of scheduling policies on memory-constrained multi-inference jobs were investigated. Besides two pre-existing used policies, also a new so-called memory aware (MEMA) scheduler was introduced. MEMA performs online scheduling based on offline gathered performance data of DNN layer loading and execution. This study shows that multi-inference jobs under stringent memory availability can benefit from resource-aware scheduling policies such as MEMA.

7 Future work

Different kinds of evaluations were left for the future for the interest of time. Evaluating the performance of more neural networks using different kinds of input would be of interest. Doing so would allow us to gain insight into how types of neural networks benefit from different scheduling policies. Furthermore, simulating different execution environments (i.e. when EdgeCaffe is one of many processes) would be considered as a valuable extension of the performed research.

Besides the exploration of different types of execution environments, also evaluation of different hardware platforms, such as ARM-based platforms, is considered to be a valuable addition. Doing so would allow us to establish whether the effects observed during this study generalize over different hardware platforms. Alternatively, testing on devices with other types of storage would allow us to better evaluate the time penalty of paging to disk during execution.

Acknowledgement

We want to express our gratitude to Dr. Lydia Chen for her time and insight. Dr. Chen's enthusiastic approach and refreshing angles of attack during the research were an invaluable for the setup and process of the research.

References

[1] 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 *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '17*, ACM Press, 2017, pp. 615–629, ISBN: 978-1-4503-4465-4. DOI: 10.1145/3037697.3037698. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3037697.3037698.

[2] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware", in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17, Niagara Falls, New York, USA: Association for Computing Machinery, Jun. 16, 2017, pp. 68–81, ISBN: 978-1-4503-4928-4. DOI: 10.1145/3081333.3081359. [Online]. Available: https://doi.org/10.1145/3081333.3081359 (visited on 04/20/2020).

[3] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices", *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017, ISSN: 1558-2590. DOI: 10.1109/MPRV.2017.2940968.

[4] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision", in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18, New Delhi, India: Association for Computing Machinery, Oct. 15, 2018, pp. 115–127, ISBN: 978-1-4503-5903-0. DOI: 10.1145/3241539.3241559. [Online]. Available: https://doi.org/10.1145/3241539.3241559 (visited on 04/21/2020).

[5] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference", in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 392–405.

[6] B. Cox, *Edgecaffe*, https://gitlab.com/bacox/edgecaffe, 2019.

[7] 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 *Proceedings of the 22nd ACM International Conference on Multimedia*, ser. MM '14, Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 675–678, ISBN: 9781450330633. DOI: 10.1145/2647868.2654889. [Online]. Available: https://doi.org/10.1145/2647868.2654889.

[8] H. Brouwer, "Modeling inference time of deep neural networks on resource-constrained systems", Jun. 2020.

[9] G. Levi and T. Hassncer, "Age and gender classification using convolutional neural networks", in *2015 IEEE Conference on Computer Vision and Pattern*

Recognition Workshops (CVPRW), IEEE, Jun. 2015, pp. 34–42, ISBN: 978-1-4673-6759-2. DOI: 10.1109/CVPRW.2015.7301352. [Online]. Available: http://ieeexplore.ieee.org/document/7301352/.

[10] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks", in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval - ICMR '15*, ACM Press, 2015, pp. 643–650, ISBN: 978-1-4503-3274-3. DOI: 10.1145/2671188.2749408. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2671188.2749408.

[11] A. Guo, *Facedetection_cnn*, 2015. [Online]. Available: https://github.com/guoyilin/FaceDetection_CNN/commits/master.

[12] J. Zhang, S. Ma, M. Sameki, S. Sclaroff, M. Betke, Z. Lin, X. Shen, B. Price, and R. Mech, "Salient object subitizing", *arXiv:1607.07525 [cs]*, Jul. 2016, arXiv: 1607.07525. [Online]. Available: http://arxiv.org/abs/1607.07525.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

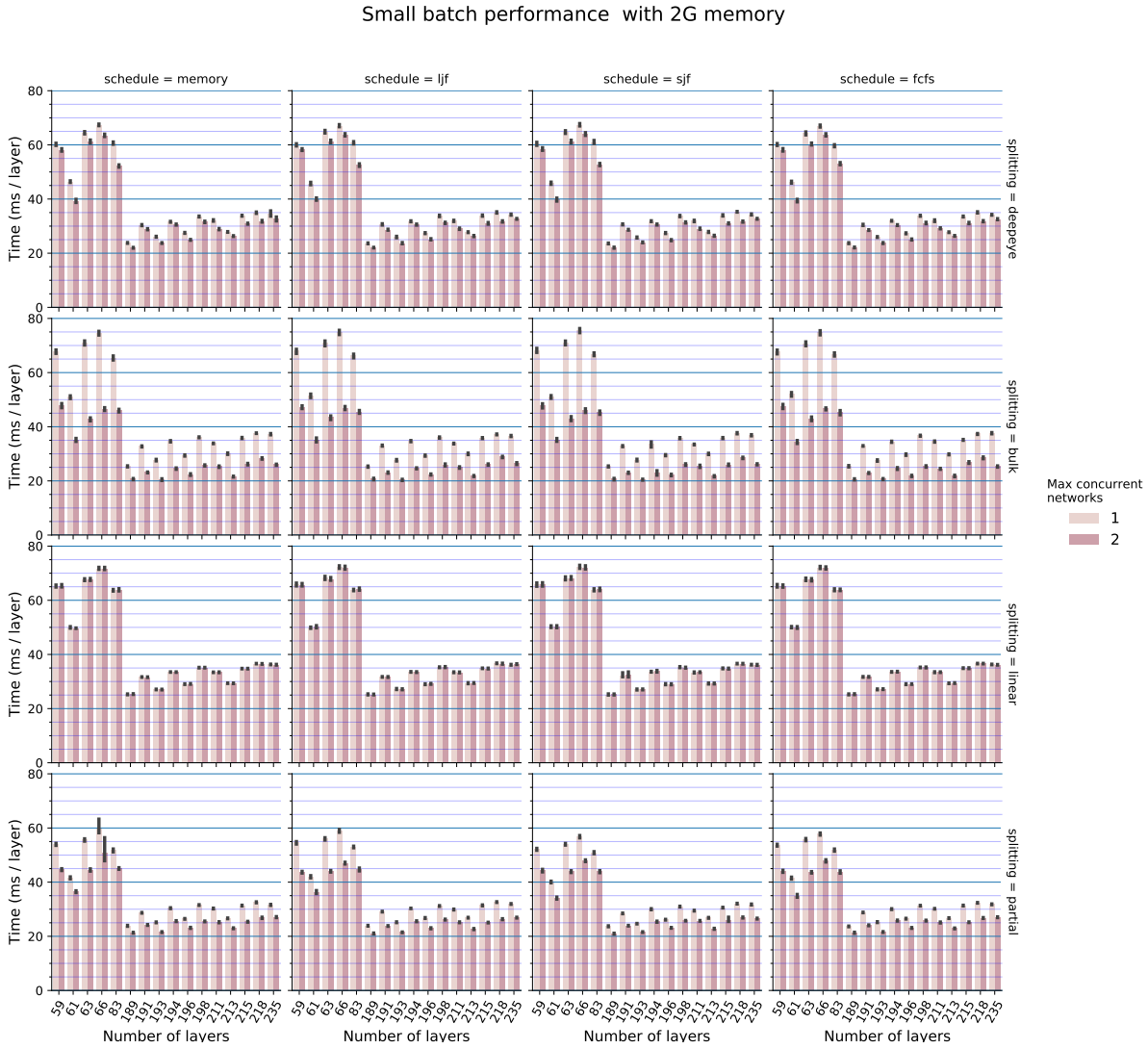
[15] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge", *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[16] D. Zhang, N. Vance, Y. Zhang, M. T. Rashid, and D. Wang, "Edgebatch: Towards ai-empowered optimal task batching in intelligent edge systems", in *2019 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, Dec. 2019, pp. 366–379, ISBN: 978-1-72816-463-2. DOI: 10.1109/RTSS46320.2019.00040. [Online]. Available: https://ieeexplore.ieee.org/document/9052125/.

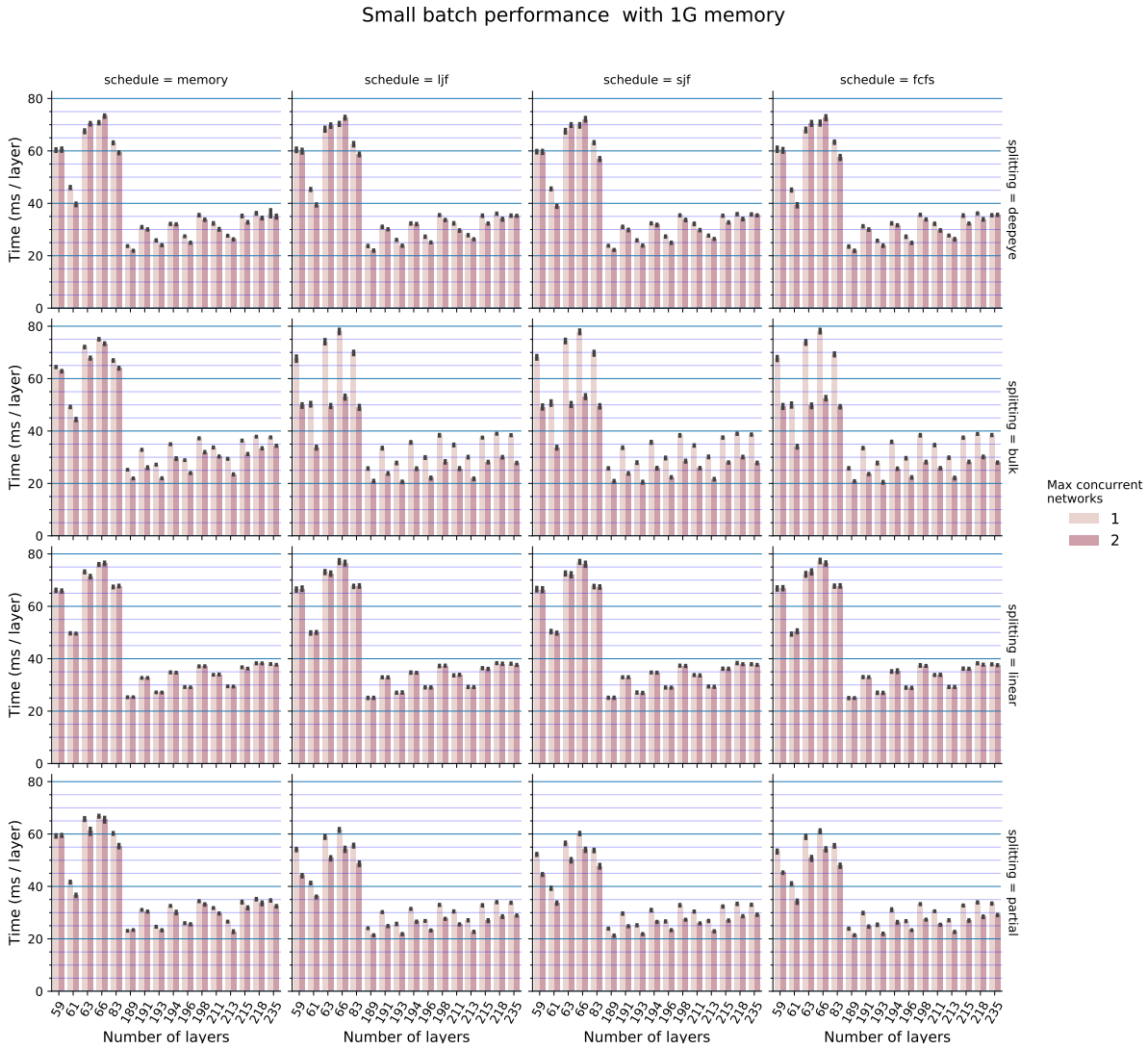
[17] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints", in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16, Singapore, Singapore: Association for Computing Machinery, Jun. 20, 2016, pp. 123–136, ISBN: 978-1-4503-4269-8. DOI: 10.1145/2906388.2906396. [Online]. Available: https://doi.org/10.1145/2906388.2906396 (visited on 04/21/2020).

[18] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018, ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2018.2858384.

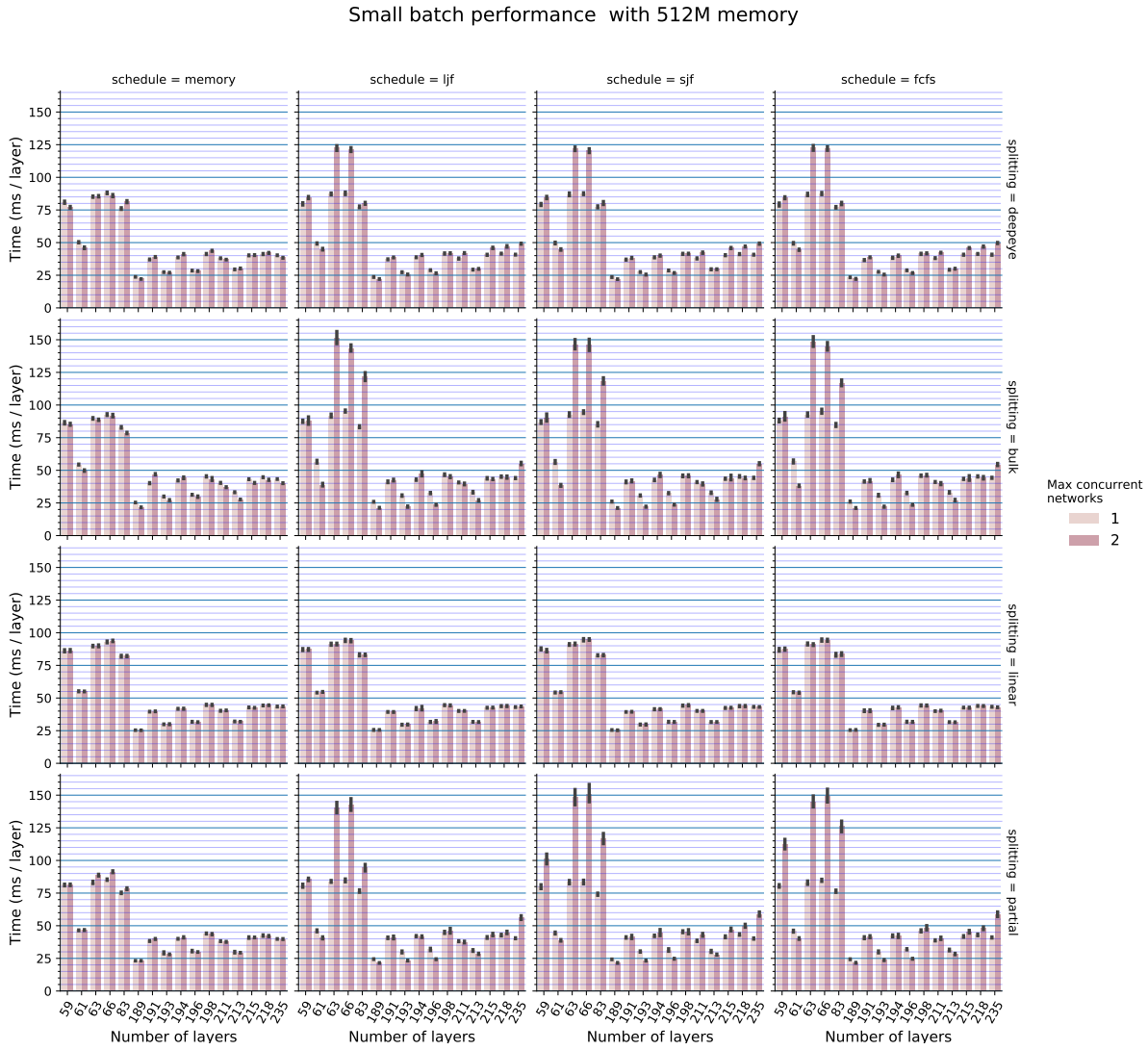
A Small batch execution performance



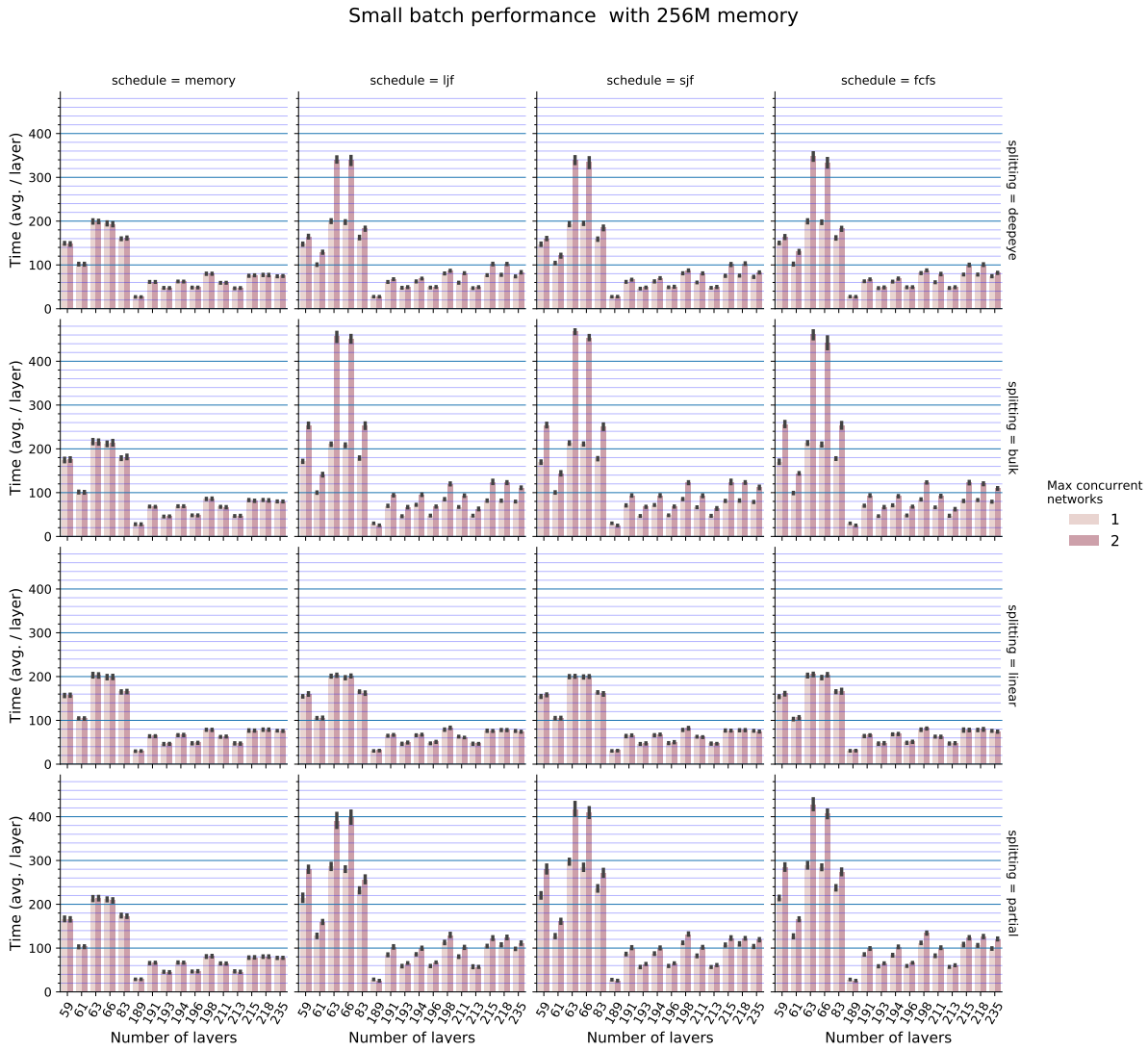
(a) Comparison of the runtime of the small batches at 1G of available memory. The scheduling policies perform comparable per splitting policy for serial execution of inference tasks, whilst the MEMA scheduler does not show a reduction in execution time when allowing concurrent execution.



(b) Comparison of the runtime of the small batches at 1G of available memory. The scheduling policies perform comparable per splitting policy for serial execution of inference tasks, whilst the MEMA scheduler does not show a reduction in execution time when allowing concurrent execution.



(c) Comparison of the runtime performance of the small batches at 512M of available memory. The greedy FCFS, SJF, and LJF policies perform comparable per splitting policy for serial execution of inference tasks. Overall an increase in execution time is visible, and a deterioration in execution speed when jobs contain larger networks. MEMA scheduling shows comparable to the other policies but does not show an increase in execution time when networks can be executed concurrently.



(d) Comparison of the runtime of the small batches at 256M of available memory. At this memory availability, all run times are increased relative to the execution with 1G available. Especially the more memory-intensive bulk is especially affected by the constraints in memory. Partial loading, which is indifferent regarding the chosen scheduling policy, performs well under this constrained memory, which performs comparably to partial loading paired with the MEMA scheduler. Due to time constraints, the evaluation on batch size 3 was not evaluated for scheduling policies other than the MEMA scheduler.

Figure 9: Execution performance graphs under varying degrees of constrained memory. Each bar represents the average execution time (seconds) of 20 independent runs. The x-axis represents the total sum of layers that was executed in a batch.