2021 IEEE International Conference on Pervasive Computing and Communications (PerCom) | 978-1-6654-0418-1/20/$31.00 ©2021 IEEE | DOI: 10.1109/PERCOM50583.2021.9439111

# MASA: Responsive Multi-DNN Inference on the Edge

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

***Abstract*—Deep neural networks (DNNs) are becoming the core components of many applications running on edge devices, especially for real time image-based analysis. Increasingly, multi-faced knowledge is extracted via executing multiple DNNs inference models, e.g., identifying objects, faces, and genders from images. The response times of multi-DNN highly affect users' quality of experience and safety as well. Different DNNs exhibit diversified resource requirements and execution patterns across layers and networks, which may easily exceed the available device memory and riskily degrade the responsiveness. In this paper, we design and implement MASA, a responsive memory-aware multi-DNN execution framework, an on-device middleware featuring on modeling inter- and intra-network dependency and leveraging complimentary memory usage of each layer. MASA can consistently ensure the average response time when deterministically and stochastically executing multiple DNN-based image analyses. We extensively evaluate MASA on three configurations of Raspberry Pi and a large set of popular DNN models triggered by different generation patterns of images. Our evaluation results show that MASA can achieve lower average response times by up to 90% on devices with small memory, i.e., 512 MB to 1 GB, compared to the state of the art multi-DNN scheduling solutions.**

***Index Terms*—Multiple DNNs inference, average response time, edge devices, memory-aware scheduling**

## I. INTRODUCTION

Edge devices, such as cameras, wearables, and sensors, are becoming an integral part of our daily life and increasingly draw on deep neural networks (DNNs) for sophisticated real-time image-based analysis. Instagram [1] enables real-time knowledge extraction directly on user's devices upon captures of images via running the *inference* of convolutional neural networks (CNNs) – one of the most widely adopted type of DNN for image processing [2], [3]. Autonomous vehicles and surveillance systems are other examples that need execute DNN inferences to recognize the surroundings and facilitate on-line decision making. It is imperative to ensure the responsiveness of DNN inference, i.e., low response time, on edge devices for the quality of experience as well as safety.

Depending on the purposes of the learning tasks, CNNs can have disparate architectures in terms of the number and size of layers. For instance, YOLO [4] can efficiently identify objects, whereas AgeNet and GenderNet [5] can discern the age and gender of people. To extract the numerous information embedded in a single image, multiple different DNN inferences need to be executed – greatly increasing the computational overhead, and the risk of resource contention as well as unresponsiveness. Moreover, images can be taken periodically or stochastically and trigger complex DNNs execution flows, e.g., GenderNet can be trigger after faces are recognized by FaceNet. We term multi-DNN inference job the analysis of the same image processed by a set of DNNs. The difficulty of maintaining low response times of both periodic and stochastically multi-DNN inference jobs is significantly higher, due to the slowdown dynamics across small and large inference networks.

DNNs are known for the high accuracy of complex tasks at the cost of intensive memory footprint and computation overhead. Typical CNN models [6], [7] can take up 892 MB memory to store 57.7 million model parameters. Executing the convolution layers of CNNs is CPU intensive, whereas computing the fully-connected layers is more memory intensive due to the high number of model weights. DNN edge inference can be achieved by using networks with a small number of layers [8], [9] but it limits the possible networks. To turn the DNN edge inference from infeasible to reality, a large body of related studies [10], [11] aims to minimize the resource demands of DNNs by compressing and pruning models for *individual* networks – achieving a calculable trade-off with the accuracy. As for multi-DNN inferences, the existing DNN frameworks, e.g., PyTorch [12], and Caffe [13], supports only execution of one model at a time. Recent studies take a step beyond single inference by considering complementary resource usages [10] or intra-network dependencies [11] without degrading network accuracy. They shed light on how multi-DNN inferences can be accelerated on devices with narrow resource availability for periodic workloads. However, stochastic workloads of multi-DNN is yet to be explored.

In this paper, we design MASA, a highly responsive and memory-aware multi-DNN execution framework. It is an on-device middleware. We aim to achieve low average response times for extracting multiple information from captured images on edge devices. To efficiently multiplex the limited CPU and memory resources across DNN models, MASA uniquely considers and models the resource demands in per layer granularities. MASA opportunistically schedules the loading and execution of each layer as to leverage complimentary resource usages of different layers, i.e., interleaving the loading and execution of convolution layers (CPU intensive) and fully-connected layers (memory intensive). To handle the challenging stochastic workloads, MASA greedily executes DNN layers by the principle of smallest memory first and then their generation orders, avoiding the performance penalty of

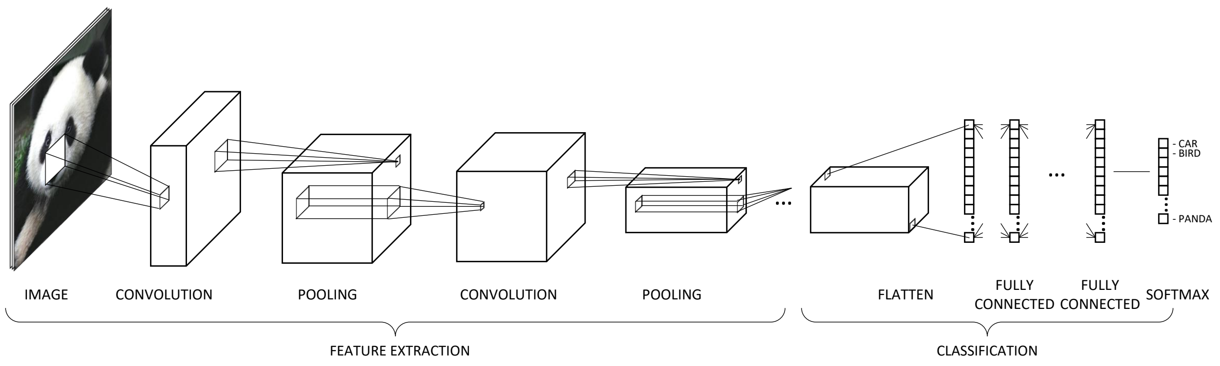


Fig. 1: Exemplary CNN architecture comprising convolutional, pooling and fully-connected layers. Flatten reshapes the input to a 1D array. Softmax performs the final classification.

memory swap. We analytically show that the memory-aware scheduling problem is probably NP-hard.

MASA consists of two key components: an offline network preparator and an online scheduler. The network preparator splits each DNN by layers and estimates the peak memory demands per layer by a light-weight offline execution. The network preparator separates layer execution into two tasks, namely memory loading, and execution. The scheduler handles both loading and execution tasks for each layer of all DNNs by dynamically checking the inter-network dependency, the available memory space, and the estimated memory demand of all available layers[1]. We implement MASA on top of Caffe, and then extensively evaluate MASA on a large set of multi-DNN inference scenarios and a representative set of hardware, namely Rasberry Pi. Our evaluation shows that the memory-aware features of MASA significantly reduce not only the response times, and resource footprints but also energy consumption, compared to the state of the art multi-DNN engines.

The contributions of MASA are multifold. First of all, MASA is a first of its kind memory-aware multi-DNN execution middleware on edge devices (Section III). Second, the scheduler of MASA can simultaneously consider complimentary resource patterns of DNN layers and handle intra-DNN dependencies when facing stochastically and deterministically generated images (Section III-C). Third, MASA is extensively proven to minimize the average response times of multi-DNN inferences for real-world applications executed on resource-constrained edge devices (Section IV).

## II. Preliminary and Motivation

In this section, we first describe the preliminary of CNN-based image analysis and the present the motivation examples to highlight the difficulty of multi-DNN inference on edge devices.

### A. Background

CNNs are one of the main types of DNNs for image-based inference. They combine convolution and pooling layers for feature extraction and complementary fully-connected layers for classification [6], [7], [14]. Fig. 1 presents an exemplary architecture. CNNs are inspired by the organization of the visual cortex. Individual neurons answer to stimuli only in a restricted region of the visual field named receptive field. Multiple such fields overlap to cover the entire visual area.

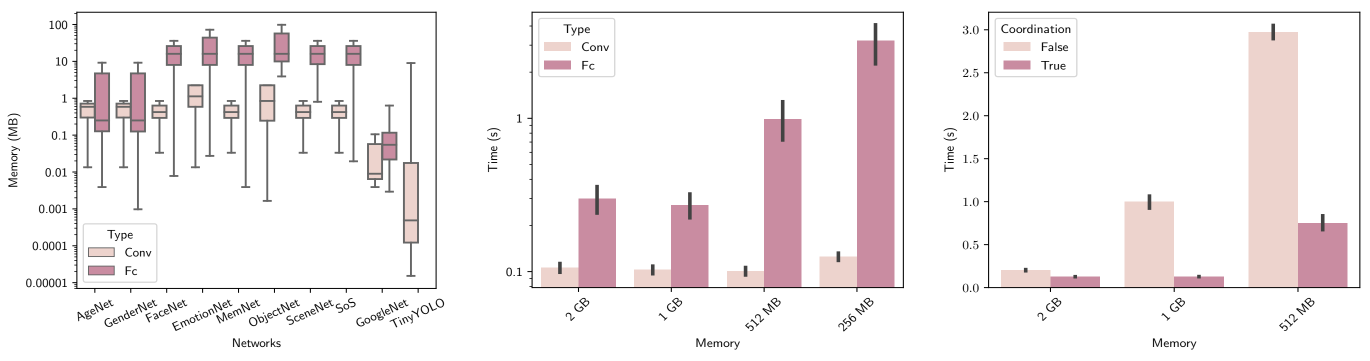
Convolutional layers divide the image in receptive fields from which they extract features via convolution with a filter matrix that is shared across receptive fields. Initial layers capture low-level features, e.g., edges and color. Subsequent layers combine low-level characteristics into higher-level features which allow the network to gain a thorough understanding of the images. Pooling layers apply aggregation functions, e.g. `max`, on fields of convolved features. This reduces noise and extracts dominant features which are positional and rotational invariant. Fully-connected layers learn non-linear combinations of the extracted high-level features and classify the image using the `softmax` classification technique. Different CNN architectures vary the number and hyper-parameters of these layers.

### B. Memory differs for networks and their layers

The runtime memory requirements of CNNs vary greatly. Traditionally DNNs are loaded in their entirety (bulk) before being executed [12], [13] favoring throughput over memory usage. Larger architectures with more and bigger layers require more memory. Table I shows examples of peak memory usage for different CNNs using Caffe (see Section IV-A for setup details). Peak memory usage ranges from 183 MB for AgeNet to 892 MB for SceneNet.

To lower peak memory requirements, especially to run large architectures on memory-constrained (edge) devices, we modify Caffe to allow fine-grained control on when layers are loaded and executed. However, memory usage can still significantly differ between layers. Fig. 2(a) summarizes the per-layer peak memory distribution for all CNNs in Table I. For each network and layer type the box shows the 25th, 50th, and 75th quartiles while the whiskers extend to the rest of the distribution. We skip pooling layers since they have negligible memory costs. For most networks, fully-connected layers have

[1]Network layers available when the image and its inference models are loaded into MASA.



(a) Peak memory distribution per layer type. (b) Diminishing available memory. (c) Lack of multi-DNN coordination.

Fig. 2: Memory usage: differences and its impact on the execution times of single- and multi-DNN inference.

one order of magnitude higher memory usage. This is due to the high number of weights, which grows quadratically with the size of the layers, i.e., equal to the product of fully-connected and its input layer sizes. Convolutional layers only define a low number of weights since the filter matrix is shared across all receptive fields. An exception to this is TinyYOLO, which only uses convolutional layers. A memory-aware scheduler needs to balance the memory usage with preloading layers for consistent performance.

## C. *Memory matters - intra-network*

The memory demands of modern CNNs can easily exceed the available memory, especially on resource-constrained edge devices, deteriorating inference responsiveness. Swapping allows OSs to execute programs with peak memory requirements that exceeds the available physical RAM by offloading memory pages to the swap space on disk. When a program accesses a memory page in the swap space a page fault is generated and the program execution is halted until the memory page has been restored to RAM. Since disk is orders of magnitude slower than RAM, programs can incur significant slowdowns based on the frequency of page faults. We demonstrate such an effect by bulk loading SceneNet as example. Fig. 2(b) shows its mean execution time split per layer type under diminishing available memory. We repeat each experiment 10 times. Error bars report the standard deviation. The peak memory usage of SceneNet is 892 MB. If the available memory is above this value, i.e., 2 GB and 1 GB RAM, execution time remains unaffected. Lower values of available memory results in longer execution time as swapping kicks in. The higher the overcommitted memory the more page faults are generated, resulting in a higher execution speed penalty. With 256 MB RAM, the execution time is 7.2× slower compared to with 1 GB RAM. From the split layer statistics, one can see that the lower the memory requirements are the lower the slowdown is because the probability of a page fault is lower. With 512 MB RAM, the fully-connected layers are already affected significantly (2.5× slowdown), while the effect on convolution layers is (still) negligible. Overall this underlines how memory-awareness matters for good performance.

## D. *Memory matters - inter-networks*

Multi-DNN inference easily exacerbates the detrimental effects of insufficient memory. Each network increases memory usage. Without coordination, this increases the probability of page faults which negatively affect execution time. We exemplify this effect by running two instances of the same network (MemNet) to ease comparison. Each instance is pinned to a separate core, i.e., the instances share memory but no compute resources. We first run the two instances in parallel (without any coordination). Then we run one instance after the other (naive coordination). We repeat each experiment 10 times. Fig. 2(c) compares the mean execution time of one MemNet instance across the two scenarios. Error bars indicate the standard deviation. Each MemNet instance uses 880 MB peak memory. When memory is sufficient, i.e., 2 GB RAM, both cases have similar execution times. The no-coordination case is slightly slower due to contention on other resources, i.e., mainly loading the weights from disk. However with multiple networks and no-coordination memory becomes a bottleneck already at 1 GB RAM. Here the no-coordination case is 7.8× slower. With 512 MB RAM, both cases are degraded but no-coordination is still 3.9× worse. Naive coordination does better but potentially misses out on complementary resource usages. This highlights the increased challenge and need for coordination when doing multi-DNN inference.

## III. MASA

We design MASA, a memory-aware multi-DNN inference framework on edge devices. MASA can efficiently process jobs composed of images and DNNs and responsively provide requested analyses, e.g., inferring faces and genders. We first describe the architecture, core components, and implementation. Then, we present the detailed memory-aware scheduling problem analysis – demonstrating its hardness.

## A. *Architecture Overview*

MASA is composed of an offline model preparator and an online scheduler as shown in Fig. 3. The offline inputs are the candidate set of trained DNN models and the online inputs are the images and specific networks.

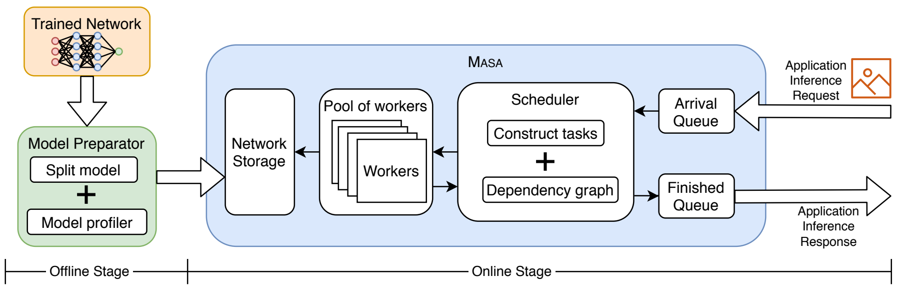


Fig. 3: Architecture of MASA.

**Model Preparator**. All trained DNN models are first processed offline by the model preparator, from model downloading, splitting, to profiling. Each DNN model gets split by layer, and their corresponding weight parameters are stored in separate files. We then profile the active memory usage of each layer by offline execution of the DNNs. The details of the profiling steps are provided in Section III-B.

**Scheduler** The scheduler is responsible for constructing two types of tasks for each network layer, namely loading model parameters from disk into memory and execution of such a layer. Herein, we abbreviate them as *loading* and *execution* tasks.

The scheduler constructs the dependency graph for tasks of each DNN and then distributes those tasks to free workers based on their dependency graph, as shown in Fig. 4. Loading tasks have to be strictly executed prior to the execution tasks of the same layers. More importantly, the scheduler dynamically monitors the memory usages and estimated memory requirement for tasks of all DNNs that are ready. The estimated memory requirement is obtained from the profiler. The detailed algorithms of the scheduler is described in Section III-C. When all tasks from a DNN are completed, the inference results are sent as a response to the application.

## B. Memory Profiler

In order to make memory-aware scheduling decisions, MASA needs accurate estimates of the memory requirements per layer. An underestimation of the memory usage causes the scheduler to over commit its memory budget and thereby likely causing paging and slowdowns. An overestimation of the memory usage can lead to under utilization of precious resources.

The memory profiler aims to precisely estimate the peak memory usage per layer which demands a finer granularity than existing well-known off-the-shelf profilers, e.g., Valgrind [15] and GPerfTools [16]. We hence resort to query the kernel via the `/proc/self/statm` interface to track the RAM usage of our process. Specifically, we execute each layer of a network in isolation by a single worker during profiling. To track the memory usage from the beginning of a layer task to its end, we spawn a parallel thread which continuously probes the consumed memory. The mean time between probing of the profiler thread is 7.12 µs $\pm 5.5$ µs, which is significantly lower than the execution time of the smallest layer considered in our evaluation. Table I lists the storage space of each network and peak run-time memory usage. One can see that the active memory usage is multiple times higher than the storage space. The detailed analysis of memory allocation is given in Section III-E.

## C. Scheduler

The scheduler sends the loading and execution tasks of all layers of DNNs that are requested for specific images to workers. It makes the memory occupancy is by a subset of layers within the space limit. The scheduler follows two principles: (i) the task dependency, i.e., loading task of layer $j$ should be completed before starting its execution task, and (ii) a hybrid order of layer type and memory constraint within and across multiple DNNs. As such, MASA incorporates the memory dependency at the inter- and intra-network levels.

The loading and execution tasks of all layers across all specified DNNs are kept sorted into two groups: the waiting and ready group. Tasks start in the waiting group. Once all their dependencies are satisfied they are moved to the ready group. Dependencies are resolved based on the dependencies graphs defined by the multi-DNN job and each DNN model. Tasks in the ready group are sorted to optimize memory consumption in a greedy fashion. The pseudocode of the scheduler is summarized in Algorithm 1. First, execution tasks are prioritized over loading tasks since they free up memory upon completion. Second, tasks are sorted by increasing memory consumption. Function *ReadyTasks()* on line 17 returns the tasks in ascending order of memory usage. When one or more workers are available, tasks are pulled from the ready group in order, checking each time if their estimated required memory exceeds the available memory. If not, the task is started via the function *Assign()* on line 10; otherwise the next task is checked. In case none of the ready tasks fits the available memory and all workers are idle, we forgo the memory constraint to allow one task to run to ensure progress (line 4). In this case we start the task with the smallest required memory of the preferred type, i.e., execution before loading.

Fig. 4 presents a scheduling example with a three-layer DNN and two workers for simplicity. The top part depicts the dependency graph between layers. The bottom part shows the scheduling timeline. Loading (blue, L) tasks must run before their corresponding execution (orange, E) tasks. Execution tasks must run in order, but loading tasks can be pre-loaded.

**Algorithm 1:** MASA scheduling algorithm

```
1  Function ValidTask(task):
2      $c \leftarrow$ BusyWorkersCount()                    // busy worker count
3      $M \leftarrow$ GetFreeMemorySpace()                  // available memory
4      return $c == 0 \lor task.req\_memory \leq M$
5  Function ScheduleTask(w, tasks):
6      for $task \in tasks$ do
7          $M \leftarrow$ GetFreeMemorySpace()              // available memory
8          if ValidTask(task) then
9              $M = M - task.req\_memory$
10             $w.Assign(task)$
11             return True
12     end
13     return False
14 Procedure MASA()
15     $W \leftarrow$ IdleWorkers()                          // set of idle workers
16     for $w \in W$ do
17         $exec,\ load \leftarrow$ ReadyTasks()            // ready tasks by type
18         if not ScheduleTask(w, exec) then
19             ScheduleTask(w, load)
20     end
21     return
```

At the start only L$_1$ can start since it does not depend on any other task. Once completed, both L$_2$ and E$_1$ are ready and executed one by each available worker. When L$_2$ completes the memory available is not sufficient to run neither E$_2$ nor L$_3$ until E$_1$ completes and frees up memory. At that point, with enough memory available, the two workers start L$_3$ and E$_2$ in parallel. Once E$_2$ completes we can start the last task E$_3$ and finish the inference.

## D. Implementation

We altered the Caffe framework to support layer-by-layer loading and execution of DNNs. This enables partial execution of DNNs and optimization of (multi-DNN) inference on edge devices. We term this new framework EDGECAFFE.

**Pool of workers** The tasks of layers are computed by workers. The scheduler takes free workers from the pool and assigns them computations. Once completed the worker returns to the pool.

**Scheduler** EDGECAFFE coordinates the work between the workers via the scheduler. The scheduler pulls multi-DNN jobs from the arrival queue, queries the model information from the network storage, builds the necessary layer handling tasks and resolve the dependency graph. This allows to assign tasks such that the correct network layers are present for computation without violating any dependency. Completed jobs are pushed to the finished queue.

**Network Storage** Trained DNNs are saved on disk in the Network Storage. A prepared network consists of splitted model files containing one or a few layers each, as well as a model description. Other components can fetch a (a subset of) model from disk though the Network Storage API by providing the model name.

**Arrival and Finish Queue** The arrival queue stores multi-DNN job requests to be processed by the scheduler. Once completed the reference results are pushed to the finish queue ready to be consumed by the application(s).

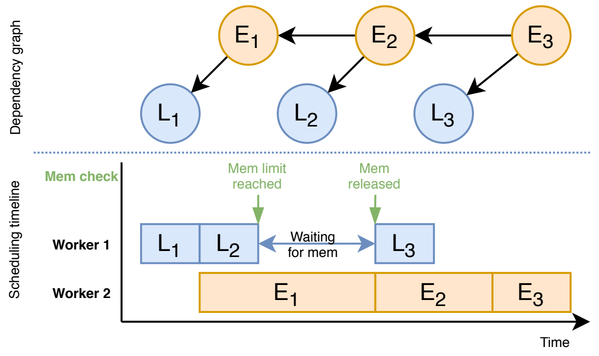


Fig. 4: MASA scheduling a 3-layer DNN on 2 workers. Networks are not fixed to workers: task $L_3$ and task $E_2$ could be executed by the other worker.

Our full implementation of EDGECAFFE and MASA is open source and available online[2].

## E. Memory Scheduling Analysis

To reason about hardness of memory aware-scheduling, we introduce SQUISHYMEMPACKING, a variant of the well known BINPACKING problem. Consider the case of a new multi-DNN inference job comprising a set of networks $\mathcal{N} = \{n_1, \ldots, n_q\}$ which must be scheduled in groups of tasks, i.e., the bins, that may not exceed a memory threshold. Each network $n_i$ is composed by $k_i$ layers. Let $\ell_{i,j}$ denote the $j^{\text{th}}$ layer of the $i^{\text{th}}$ network. Each layer comprises two tasks which need to be scheduled: one to load the layer into memory and one to compute and store the output. Let $m^{\mathcal{L}}(\ell)$ and $m^{\mathcal{E}}(\ell)$ denote the memory required to load, and compute and store the output of a network layer.

Our framework allows to schedule networks at layer granularity, but layers of the same network have strict dependency requirements given by their ordering. Hence, we define a bin $\mathcal{B}$ as a set of layers to be scheduled together comprising ranges of layers of different networks, i.e., $B = \{\mathcal{B}_g, \ldots, \mathcal{B}_h\}$, where $\mathcal{B}_i = \{\ell_{i,j} \,|\, a < j \leq b\}$ is the range of layers between $a$ and $b$ belonging to network $n_i$ with $0 \leq a < b \leq k_i$. Whereas the loading memory of layers is static, the memory needed for computation of intermediate layer results is re-usable by next layers. The needed memory to execute the layers $\mathcal{B}_i$ of a given network $n_i$ is partially *squishy*, depending on the computation memory required by the layers in $\mathcal{B}_i$. Hence, skipping the input, the memory required to run a $\mathcal{B}_i$ is as follows.

$$\sum_{\ell \in \mathcal{B}_i} m^{\mathcal{L}}(\ell) + \max_{\ell \in \mathcal{B}_i} \left\{ m^{\mathcal{E}}(\ell) \right\} \quad (1)$$

Once a bin $\mathcal{B}$ is completed, memory allocated for loaded layers can be released, as well as the memory reserved for the execution of finished networks. As such, let us define $\mathcal{N}_a = \{n_i \in \mathcal{N} \,|\, n_i \textit{ has been started in a previous bin.}\}$, and $\mathcal{N}_b = \{n_i \in \mathcal{N} \,|\, n_i \textit{ will start in a next bin.}\}$. Additionally, let $m^{\mathcal{E}}_{i,o}$ be the memory used by the last executed layer of network $n_i$,

[2]https://github.com/bacox/edgecaffe

TABLE I: Overview of used DNNs in the experimental setup.

| Network[3] | Inference | Storage Space(MB) | Memory Usage(MB) | Layers conv | Layers fc | Layers total |
|---|---|---|---|---|---|---|
| AgeNet [5] | Age | 44 | 183 | 3 | 3 | 19 |
| GenderNet [5] | Gender | 44 | 186 | 3 | 3 | 19 |
| FaceNet [17] | Face | 217 | 875 | 5 | 3 | 23 |
| SoS [18] | Saliency | 218 | 875 | 5 | 3 | 21 |
| GoogleNet [18] | Saliency | 23 | 404 | 22 | 2 | 151 |
| TinyYOLO [4] | Object | 62 | 263 | 9 | - | 39 |
| EmotionNet [19] | Emotion | 378 | 761 | 5 | 3 | 22 |
| MemNet [20] | Memorability | 217 | 880 | 5 | 3 | 22 |
| SceneNet [21] | Object | 221 | 892 | 5 | 3 | 23 |

or zero when the network has not been started yet. As such the memory requirement for a set of tasks becomes

$$\sum_{n_i \in \mathcal{N}_a \setminus \mathcal{N}_b} m^{\mathcal{E}}_{i,o} + \sum_{n_i \in \mathcal{N}_b} \Big( \big( \sum_{\ell \in \mathcal{B}_i} m^{\mathcal{L}}(\ell) \big) + \max_{\ell \in \mathcal{B}_i} \{ m^{\mathcal{E}}(\ell), m^{\mathcal{E}}_{i,o} \} \Big) \tag{2}$$

It can be easily shown that BINPACKING is reducible in polynomial time to SQUISHYMEMPACKING by equaling each element $i \in I$ for a BINPACKING instance to a single layer network, such that the size is set to the loading memory and no execution memory overhead, i.e., size $s(i) = m^{\mathcal{L}}_{i,1}$ and $m^{\mathcal{E}}_{i,1} = 0$. Therefore, as SQUISHYMEMPACKING is NP-hard, we argue that finding an optimal solution is not feasible, and a heuristic as MASA is a good alternative.

## IV. EVALUATION

This section presents our in-depth evaluation of MASA using real-world DNN applications on representative edge devices. Our main findings are:

1) MASA can consistently meet the deadlines for periodic multi-DNN inference jobs. The performance gain is particularly prominent on smaller memory devices and highly heterogeneous multi-DNN jobs.
2) MASA can effectively interleave the execution/loading of convolution and fully-connected layers to minimize the response times and energy consumption, especially for stochastically generated images.
3) The efficacy of MASA is robust to errors in memory estimation. Moreover, it incurs low computation and memory overhead on edge devices.

### A. Experimental setup

**Edge Devices**. We consider RaspberryPi (termed RPi) as representative edge devices because of its wide adaptability and ease of programming. Specifically, we select three configurations of RPi: (i) RPi 3B+ equipped with Cortex-A53 (1.4 GHz) and 1 GB memory, (ii) RPi 4B with Cortex-A72 (1.5 GHz) and 2 GB memory, and (iii) RPi 4B with Cortex-A72 (1.5 GHz) and 4 GB memory. To emulate the scenario of multiple applications on edge devices, we only consider memory sizes of 512 MB, 1 GB and 2 GB in the following evaluations. Each RPi is equipped with a SanDisk Extreme 64 GB microSD card for storage.

**Multi-DNN Jobs**. We consider 9 types of DNNs, namely AgeNet, GenderNet, FaceNet, SoS, GoogleNet, TinyYOLO, EmotionNet, MemNet, and SceneNet, to analyze images from the EDUB-Seg dataset [22], [23]. We note that as the models are not altered, e.g., not compressed or pruned, the accuracy of the network is not impacted. Each network conducts various kinds of image analysis, e.g., inferring age, gender, faces, and salient objects, and differ in their network structures and sizes. We summarize all networks considered in Table I, where disk space and active memory usage obtained from our profiler are listed. EmotionNet is the largest network in terms of absolute storage space, i.e., in the order of 380 MB. SceneNet is the largest network in terms of memory usage, i.e., in the order of 890 MB. Networks listed in Table I are commonly used for multi-DNN inference applications like lifelogging [10].

We emulate scenarios of multi-DNN inference by executing multiple of such DNN models on periodically and stochastically generated images. The specific composition and number of DNN models are given in each subsection. For periodical case, single images generate at a fixed interval. For stochastic case, both inter-generation times and number of DNNs per image follow normal and uniform distribution, respectively.

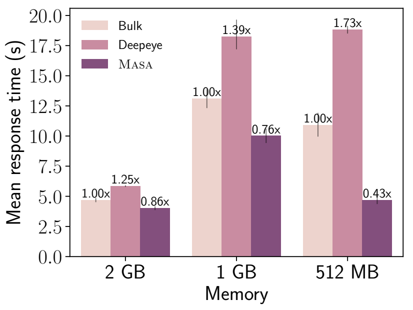
**Performance Measures**. We aim to optimize the average response times per multi-DNN job. The response time is measured from the moment the image generates till the time all DNN models have completed execution. For periodical generation, we also set deadlines which are imperative for safety critical systems. All values presented in the following sections are averaged over 150 images and more than 450 DNN inferences.

**Comparison**. We compare MASA against `Bulk` and `DeepEye` [10]. `Bulk` executes one DNN at a time by first loading all layers at once and then executing them. `DeepEye` interleaves the execution and loading of convolution and fully-connected layers. Different from MASA, `DeepEye` acts on the granularity of layer type, instead of individual layers and is not aware of their memory usages by loading one type of network layers at once. We implement both approaches on EDGECAFFE and set two worker threads.

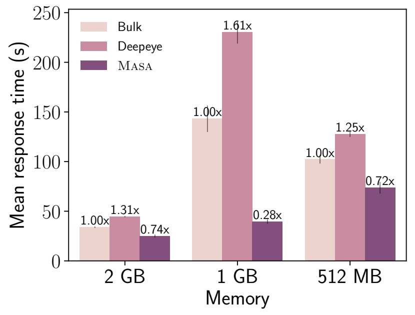
### B. Deterministic Image Analysis

We consider three types of workload scenarios that analyze periodically images: (i) three small DNN models: AgeNet, GenderNet and TinyYOLO, (ii) three mixed DNN models both small and large: AgeNet, EmotionNet, and FaceNet, and (iii) lifelogging scenario where 5 DNNs are used to automatically annotate captured images: TinyYOLO, EmotionNet, MemNet, SceneNet, and SoS. The DNN models used in the lifelogging scenario are typical networks used to give useful annotations to images captured during a lifelogging activity [10]. The images are generated every 20, 200, 200 seconds for small, mixed and lifelogging scenarios, respectively, and the analysis of multi-DNN needs to be completed before the generation of the next job.

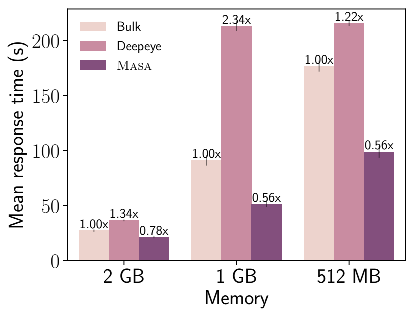
We summarize all three scenarios in Fig. 5. For small and mixed networks, MASA achieves the lowest response time, trailed by `Bulk` and `DeepEye` across all combinations of workload scenarios and RPi configurations, as shown in



(a) Small networks



(b) Mixed networks



(c) Lifelogging

Fig. 5: Average response times of deterministic image generations: comparisons of `Bulk`, `DeepEye` and MASA.

Fig. 5(a) and Fig. 5(b). The performance of `DeepEye` often has the highest response times because multiple DNNs are greedily loaded into the memory and cause costly memory swap operations. Meanwhile, due to the extra consideration of intra-network dependency, MASA outperforms `Bulk` by at least 30 %, in case of 512 MB memory.

Another observation worth mentioning is the trend of performance gain of MASA. It has significant performance gains on devices with smaller memory, i.e., 512 MB, whereas the performance gain on 2 GB memory is less significant. Moreover, the relative performance gains of MASA against other approaches is the highest on small devices and homogeneous DNNs, compared to large devices and mixed workloads. This can be explained by the balanced task times of execution and loading and avoidance of memory swapping.

In Fig. 5(c), we summarize the average response time of the lifelogging application that emulates a real-life application, executing five DNN inferences for every captured image. Similar trends as for the previous two scenarios can be observed: MASA can effectively allocate the limited worker threads and memory to achieve the lowest response time. The average response time of MASA in the case of 1 GB memory is similar to `Bulk` and `DeepEye` running on 2 GB memory. In other words, MASA can achieve almost 50% resource saving without degrading response times.

## C. *Stochastic Image Analysis*

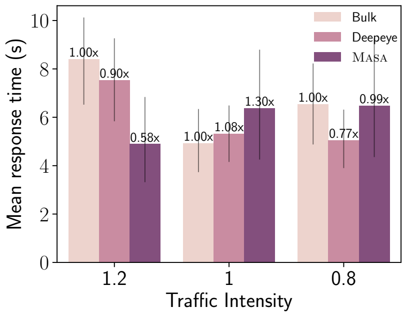
Here, we consider stochastic image analyses, where either the image generations or the number of DNN inferences, or both are generated stochastically. This workload scenario is much more complex than the periodic cases due to the intricate intra-network dependency and heterogeneity of inference jobs.

Specifically, two types of stochastic scenarios are evaluated with increasing stochasticity. (i) Scenario I: single DNN inference randomly drawn from the nine listed in Table I is requested upon the generation of images following normal distribution where the mean is the sum of the average execution of the candidate models and a standard deviation of 200 ms, respectively. Due to the high variance of inter-generation times, multiple DNNs need to be processed at the same time. (ii) Scenario II: the images arrive periodically but the composition of multiple DNNs is randomly requested following a uniform distribution. For each scenario, we evaluate MASA on three memory configurations, and three traffic intensities. We normalize the traffic intensity with respect to the mean of the execution time of all networks in the scenario, as such values of 0.8, 1.0 and 1.2 are evaluated. The higher the value of traffic intensity becomes, the more challenging it is to achieve low response times.

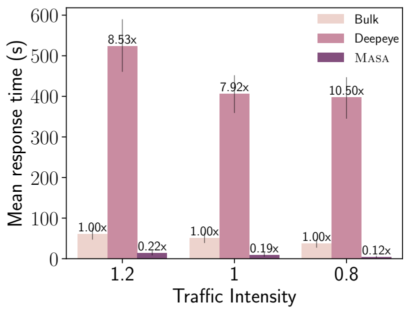
Fig. 6 and Fig. 7 summarize the average response times of the aforementioned stochastic scenarios and the relative performance gain with respect to `Bulk` and `DeepEye`. Similar trends as the deterministic case can be observed: MASA achieves the lowest response times for small memory and high traffic intensity. In terms of absolute values, here the relative gains are significantly higher than for the deterministic scenarios which only need to cope with predictable workloads.

**Scenario I**. In Fig. 6, one can clearly see that the performance gains of MASA against `Bulk` and `DeepEye` are more pronounced for higher traffic intensity and lower memory. In the cases of 1 GB and 512 MB memory (shown in Fig. 6(b) and Fig. 6(c)), we reduce the average response times up to 90%, compared to the second best policy. In the case of 2 GB memory shown in Fig. 6(a), we are slightly worse especially for the lighter traffic intensity. This can be explained by the fact that under light traffic greedy loading algorithms like `Bulk` and `DeepEye` are better to recover from the impact of memory paging than under heavy traffic. Moreover, we would like to point out that the average response times of `Bulk` and `DeepEye` increase from around 6 up to several hundreds seconds when memory space is reduced from 2 GB to 512 MB. In contrast, MASA can show only a 1.5X increase of the response times relative to the memory reduction, i.e., from 5 seconds at 2 GB up to 60 seconds at 512 MB.

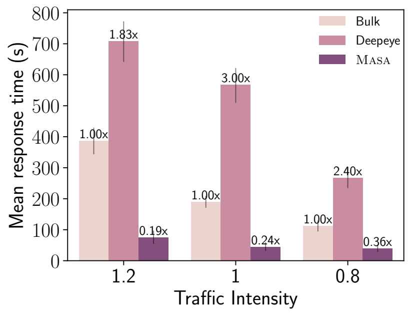
**Scenario II**. In Fig. 7 we can see that MASA is still the best performing policy, trailed by `DeepEye` then `Bulk`. As this scenario is slightly more predictable than scenario I due to the periodical image generations, the performance gain is slightly lower than scenario I. `DeepEye` is able to manage the response time by greedily interleaving the model loading and execution. Here, MASA can maintain close to superlinear ratio between the average response times and available memory, i.e.,



(a) 2 GB memory

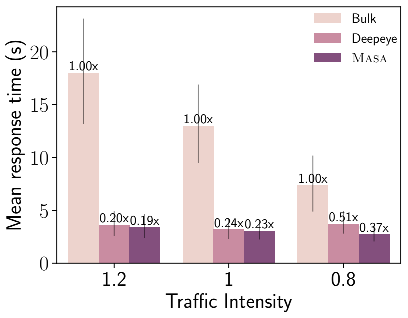


(b) 1 GB memory

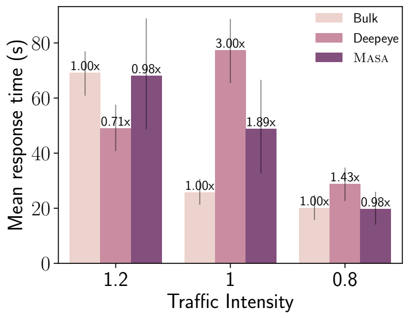


(c) 512 MB memory

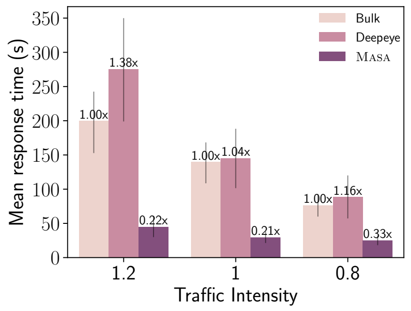
Fig. 6: Average response times of stochastic scenario I: comparisons of `Bulk`, `DeepEye` and MASA.



(a) 2 GB memory



(b) 1 GB memory



(c) 512 MB memory

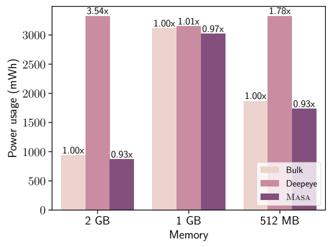
Fig. 7: Average response times of stochastic scenario II: comparisons of `Bulk`, `DeepEye` and MASA.

from 3 seconds at 2 GB up to 30 seconds at 512 MB at a traffic intensity of 1.

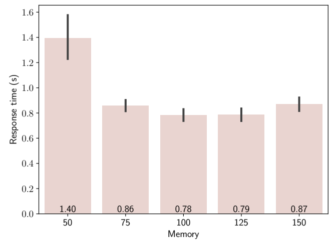
MASA is able to achieve such remarkable results due to its intelligent memory management and interleaved execution of DNN layers. We also evaluate the effectiveness of MASA on different number of worker threads. MASA can robustly ensure low response times, whereas `Bulk` and `DeepEye` can not properly take advantage of multiple worker threads. Increasing the number of workers to a value larger than 2 allows MASA to better interleave tasks when multiple DNNs are executed concurrently. This flexibility in scaling is not seen in `Bulk` and `DeepEye`. For example, doubling the number of workers from 2 to 4 under a random high traffic intensity and 3 small DNNs, MASA reduces the mean response time by 79% compared to 15% and 9% achieved by `DeepEye` and `Bulk`, respectively. Due to the space limit, we skip the detailed presentation of such results.

## D. Energy Consumption and Overhead

In Fig. 8(a), we present the energy consumption of `Bulk`, `DeepEye`, and MASA, obtained by deterministically executing three small DNN inferences (details in Section IV-B). As the on-board tools of the RPi are unfit for accurately measuring power, an external power monitor (Joy-IT JT-TC66C) is used that measures the voltage and current drawn from the power supply. We measure the absolute time the edge devices are active and the corresponding energy consumption in terms



(a) Power usage [mWh] when executing 150 three small DNN.



(b) The effect of under- and over-estimation of memory needed.

Fig. 8: Power and sensitivity analysis

of mWh. The energy values are on a par with the measured response times. As a result, the energy consumption of MASA is the lowest due to its shorter time to process all 150 images. MASA is explicitly designed to increase the responsiveness of DNN inference on edge but indirectly reduces the energy consumption too.

Additionally we measure the computation overhead incurred by three frameworks. To such an end, we execute 1000 dummy DNNs where each task needs zero execution time. This way, the measured response times equal to the processing overhead of each policy. The time overheads of executing each DNN are 19.0, 19.9, and 22.0 ms for `Bulk`, `DeepEye` and MASA, respectively. The slightly higher overhead of MASA is due to the fact it needs to keep track of the memory usage during execution.

### E. *Sensitivity Analysis of Memory Estimate*

Here, we test the robustness of MASA in response to inaccurate memory estimates. To such an end, we multiply the profiled memory requirements of all layers in AgeNet by 0.5 and 0.75 to show under-estimation and by 1.25 and 1.5 to show over-estimation of memory. We run a batch of 6 AgeNet networks with a repetition of 20 image generations. We summarize the results in Fig. 8(b). The under-estimated memory values cause the algorithm to load too many networks at the same time while the over-estimated memory values cause the algorithm to under-utilize the available memory. In the case of severely underestimation (the 0.5 multiplier), the average response times increase from 0.8 to 1.4 seconds as well as the standard deviation. Overall, conservative estimation of memory (even up to 50% higher than actual values) only incur small percentage increases of response times, e.g. 11.5% for 50% overestimation.

### F. *Limitations*

MASA assumes that the trained model can be split into independent stages, e.g. DNN layers, which fit the available memory. Models that cannot be split or have layers that are much larger than the available memory, will have a lowered response time performance. Another limitation are models where the input of a layer depends on a large number of preceding layers. This will result in an increase in peak memory usage of stages which can again reduce the effectiveness of MASA.

## V. RELATED WORK

We compare MASA with state-of-the-art DNN inference frameworks under two aspects: being memory aware and their focus on multi-DNN inference. The detailed comparison is summarized in Table II with additional emphasis on the evaluated workload scenarios and hardware architectures.

TABLE II: Overview of prior-art.

| **Method** | **Multi-DNN** | **Stochasticity** | **No DNN Similarity** | **Memory aware** | **Architecture** GPU | CPU |
|---|---|---|---|---|---|---|
| **MCDNN [24]** | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ |
| DeepEye [10] | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ |
| NeuOS [11] | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ |
| NestDNN [25] | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ |
| DeepMon [26] | ✗ | ✗ | – | ✓ | ✓ | ✗ |
| PatDNN [27] | ✗ | ✗ | – | ✓ | ✓ | ✓ |
| DeepCache [28] | ✗ | ✗ | – | ✓ | ✓ | ✓ |
| DART [29] | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ |
| **MASA** | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |

### A. *Memory-aware DNN Inference*

As the size and complexity of DNN applications have grown, their need for computing resources and memory has increased tremendously. It is particularly vital to manage and control the execution of DNNs on edge devices that have limited RAM. Existing solutions resort to model compression [30], [31], quantization. [32] , or network pruning [27], [30] to reduce the memory demands of DNNs. DeepMon [26] and DeepCache [28] are mobile deep learning inference frameworks which accelerate CNNs execution using cache mechanisms for processing convolutional layers. Jiang et al. [32] use graph optimizations and quantization to increase the performance of CNNs in terms of inference speed. Yang et al. [30] propose a mixed pruning method with a minimal accuracy penalty, which reduces the number of parameters in a DNN by removing redundant layers. PatDNN [27] is a pattern-based DNN pruning framework, including compressed weight storage, register load redundancy elimination, and parameter auto-tuning.

The common theme of aforementioned approaches is to tradeoff the accuracy for the resource efficiency for a single DNN, whereas MASA manages the memory requirements of multiple DNNs without altering the network structure and its resource demands.

### B. *Multiple DNNs*

Running multiple models on the same device at the same time or sharing resources with other processes is an effective but challenging way to multiplex the limited available resources. Several recent studies [10], [11], [25], [33], [34] propose solutions to efficiently run multi-DNN inference such the energy consumption, response time and accuracy can be optimized. NeuOS [11] minimizes a three-dimensional space, including latency, accuracy, and energy at the layer granularity for multi-model execution in autonomous systems. DeepEye [10] optimizes the execution of multiple CNN models by interleaving the execution of convolutional execution and fully-connected layers for memory-limited wearable devices. The study in [33] dynamically determines which DNN should be selected to execute a given input by considering the desired accuracy and inference time. NestDNN [25] allows for models to run concurrently and scale down to a model with smaller resource demands by using multi-capacity models and surrendering accuracy. MCDNN [24] relies on model sharing of variants with the same base model to efficiently run the same model type with different tasks. DART [29] is a pipeline-based real-time scheduling framework with data parallelism. DART groups a subset of DNN layers (task-level stage) and assigns them to workers of CPUs and GPUs to minimize the task response times by balancing tasks over resources while respecting time constraints. MemA [35] compares the effect of the execution time of a multi-DNN inference in the context of different scheduling algorithms. While yielding promising results, the solution proposed in MemA has difficulties to adapt to unexpected changes in the available memory space. Existing scheduling techniques such as constraint progamming to find an exact solution are NP-hard and are not scalable [36].

The aforementioned studies accelerate the inference time of multi-DNNs by leveraging the similarity across DNNs to reduce the computation, without being aware of their active memory usages. Different from them, the proposed MASA dynamically schedules DNNs according to the active memory status and does not rely on the similarity of DNN structures.

## VI. Conclusion

Motivated by the importance and emerging trend of conducting real-time image analyses at edge devices, we design and implement MASA, a responsive memory-aware multi-DNN execution middleware. MASA carefully models and manages the run-time memory usage of convolutional and fully-connected layers of all DNNs. As such, it leverages the complimentary CPU/memory usage patterns of layers within and across networks to efficiently schedule multiple DNN inferences simultaneously. We evaluate MASA on comprehensive workload scenarios, i.e., combinations of 9 different DNNs on three hardware configurations of Rasberry Pi. Our results show that MASA can significantly reduce the average response times of multi-DNN inferences by up to 90%, compared to state-of-art multi-DNN solutions. MASA is particularly effective for challenging scenarios where the available memory is low, e.g., 512 MB to 1 GB memory, and multiple DNN inferences are conducted on stochastically captured images. For future work, we will focus on improving the energy efficiency and extend MASA to applications with different and mixed types of DNNs.

## VII. Acknowledgements

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

## References

[1] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *IEEE HPCA*, pp. 331–344, 2019.

[2] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Mathematics and Computers in Simulation*, vol. 177, pp. 232 – 243, 2020.

[3] V. A. Sindagi and V. M. Patel, "A survey of recent advances in cnn-based single image crowd counting and density estimation," *Pattern Recognit. Lett.*, vol. 107, pp. 3–16, 2018.

[4] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *CVPR*, pp. 7263–7271, 2017.

[5] G. Levi and T. Hassncer, "Age and gender classification using convolutional neural networks," in *IEEE CVPRW*, p. 34–42, 6 2015.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, pp. 1106–1114, 2012.

[8] M. Abbas, M. Elhamshary, H. Rizk, M. Torki, and M. Youssef, "Wideep: Wifi-based accurate and robust indoor localization system using deep learning," in *IEEE PerCom*, pp. 1–10, 2019.

[9] K. Ochiai, F. Putri, and Y. Fukazawa, "Local app classification using deep neural network based on mobile app market data," in *IEEE PerCom*, pp. 186–191, 2019.

[10] 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.

[11] 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.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NIPS*, pp. 8026–8037, 2019.

[13] 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 Multimedia*, pp. 675–678, 2014.

[14] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[15] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *ACM SIGPLAN* (J. Ferrante and K. S. McKinley, eds.), pp. 89–100, 2007.

[16] Google, "Google Performance Tools." https://github.com/gperftools/gperftools, 2011.

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

[18] J. Zhang, S. Ma, M. Sameki, S. Sclaroff, M. Betke, Z. Lin, X. Shen, B. Price, and R. Mech, "Salient object subitizing," in *IEEE CVPR*, pp. 4045–4054, 2015.

[19] G. Levi and T. Hassner, "Emotion recognition in the wild via convolutional neural networks and mapped binary patterns," in *ACM ICMI*, pp. 503–510, 2015.

[20] A. Khosla, A. S. Raju, A. Torralba, and A. Oliva, "Understanding and predicting image memorability at a large scale," in *IEEE ICCV*, pp. 2390–2398, 2015.

[21] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *NIPS*, pp. 487–495, 2014.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] Z. Jiang, T. Chen, and M. Li, "Efficient deep learning inference on edge devices," in *ACM SysML*, 2018.

[33] 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.

[34] 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.

[35] J. Galjaard, B. Cox, A. Ghiassi, L. Y. Chen, and R. Birke, "Mema: Fast inference of multiple deep models," in *IEEE PerCom*, p. to appear, 2021.

[36] B. Cox, "Multi-model inference on the edge: Scheduling for multi-model execution on resource constrained devices," Master's thesis, TU Delft, the Netherlands, 2020.