

Performance Analysis and Characterization of Training Deep Learning Models on Mobile Devices

Jie Liu, Jiawen Liu, Wan Du and Dong Li

University of California, Merced

{jliu279, jliu265, wdu3, dli35}@ucmerced.edu

Abstract—Training deep learning models on mobile devices recently becomes possible, because of increasing computation power on mobile hardware and the advantages of enabling high user experiences. Most of the existing work on machine learning at mobile devices is focused on the inference of deep learning models, but not training. The performance characterization of training deep learning models on mobile devices is largely unexplored, although understanding the performance characterization is critical for designing and implementing deep learning models on mobile devices.

In this paper, we perform a variety of experiments on a representative mobile device (the NVIDIA TX2) to study the performance of training deep learning models. We introduce a benchmark suite and a tool to study performance of training deep learning models on mobile devices, from the perspectives of memory consumption, hardware utilization, and power consumption. The tool can correlate performance results with fine-grained operations in deep learning models, providing capabilities to capture performance variance and problems at a fine granularity. We reveal interesting performance problems and opportunities, including under-utilization of heterogeneous hardware, large energy consumption of the memory, and high predictability of workload characterization. Based on the performance analysis, we suggest interesting research directions.

I. INTRODUCTION

Deep learning models have been widely deployed on mobile devices (e.g., mobile phones and smart home hub) to process on-board sensing data and enable a variety of mobile applications (e.g., machine translation, speech recognition, cognitive assistance and street navigation) [1]–[3]. Those models are deployed for model inference (not for model training). The existing work has been conducted to analyze the performance and resource utilization of deep learning workloads on mobile devices when those models are deployed for model inference [4]–[10]. Those studies are important for optimizing the performance of deep learning models on mobile devices.

Besides model inference, training deep learning models on mobile devices recently becomes possible, because of increasing computation power on mobile hardware and the advantages of enabling high user experiences. In particular, training deep learning models opens up a new approach to utilize the computational resources. As the hardware of mobile devices is increasingly powerful and domain-specific, especially with the emergence of artificial intelligence (AI) chipsets and powerful mobile GPU [11]–[14], training deep learning models is moving from the cloud to mobile devices to leverage these decentralized computational resources. Furthermore, training deep learning models on mobile devices can

avoid transmitting user data to the cloud as in the traditional method. The traditional method can cause the breach of user privacy (even using an anonymous dataset and mixing it with other data). For applications where the training objective is specified on the basis of data available on each mobile device, training on mobile devices can significantly reduce privacy and security risks by limiting the attack surface to only the device.

Most of the existing work on machine learning at mobile devices is focused on the inference of deep learning models (particularly convolutional neural network (CNN) and recurrent neural network (RNN)), but not training. The performance characterization of training deep learning models on mobile devices is largely unexplored, although understanding the performance characterization is critical for designing and implementing deep learning models on mobile devices.

The recent work studies the performance of training deep learning models on servers [15]. However, training on mobile devices and on servers have different requirements, and have to be studied separately. First, training deep learning networks should not interfere with the user’s regular operations on mobile devices; The interference can manifest as unexpected shorter battery life because of large energy consumption caused by training deep learning networks, or the extended latency of the user’s operations. Second, the training data is likely to be collected and used for training on a daily basis. For example, to train a deep learning network for image classification, the system can use images collected per day as training samples and train the model every day. Third, a mobile device usually has a small memory capacity (compared with servers), hence some large deep learning models with tens of GB memory footprint (e.g., ResNet201 and VGG19) are not suitable to be trained on mobile devices.

In this paper, we perform a variety of experiments on a representative mobile device (the NVIDIA TX2) to study the performance of training deep learning models. Our study provides insightful observations and reveals potential research opportunities. In particular, this paper aims to answer the following research questions.

First, is training a deep learning network on a mobile device even possible? The existing work on federated learning has shown preliminary success of training some machine learning models on mobile devices [16]–[26]. Different from a typical deep learning model, those machine learning models are small in terms of memory consumption and model size. Training deep learning models is known to be compute-

intensive and memory-intensive. Traditionally deep learning models are trained on servers with GPU with thousands of cores and high memory bandwidth. However, mobile devices are under recourse constraint, e.g., limited computation power and relatively small memory capacity. It is unknown whether and which deep learning models are trainable.

Second, how does training various deep learning models in mobile devices differ? Deep learning models have shown success in a broad range of application domains. Many deep learning models, such as DenseNet, Inception, ResNet, SqueezeNet and XceptionNet are related to image classification, which is one of the most common application domains for deep learning models. Other kinds of deep learning models, such as reinforcement learning, Generative Adversarial Network (GAN) that are used in other application domains such as robot controls, image generation and natural language processing, should also be taken into consideration. We aim to explore a diverse set of deep learning models in our study.

Third, what are the major performance problems when we train deep learning models on mobile devices? Are those problems on mobile devices different from those on servers? Answering the two questions is useful to identify research problems and train deep learning models more efficiently on mobile devices.

By conducting extensive experiments with various deep learning models on a specific mobile device, the NVIDIA TX2, we find many insightful observations. In summary, we make the following contributions.

- We introduce a benchmark suite for studying the workload of training deep learning models on mobile devices. The benchmark suite includes four application domains and includes ten common deep learning models. Those benchmarks are chosen with the consideration of possible resource constrained on mobile devices. We make our benchmark suite open-source and intend to continually expand it to support various mobile devices.
- We introduce a tool to study performance of training deep learning models on mobile devices, from the perspectives of the memory consumption, hardware utilization, and power consumption. More importantly, the tool can correlate performance results with fine-grained operations in deep learning models, providing capabilities to capture performance variance and problems at a fine granularity.
- We reveal interesting performance problems and opportunities, including under-utilization of heterogeneous hardware, large energy consumption of memory, and high predictability of workload characterization. Based on the performance analysis, we suggest interesting research directions.

II. TRAINING DEEP LEARNING MODELS ON MOBILE DEVICES

Deep learning is a general-purpose method that can be used to learn and model complicated linear and non-linear relationships between input datasets and output. Many deep learning models can be represented as a directed acyclic graph

where nodes of the graph are connected neurons. Embedded in the graph, there are a number of parameters (e.g., “weights” and “bias”). Those neurons and parameters are organized as layers. The process of obtaining the optimal values of those parameters to reach high prediction accuracy is called “training”. Training involves many iterations of computation (sometimes millions of iterations), in order to acquire the optimal parameters. Each iteration is a *training step*, and consists of forward and backward passes. During the backward pass, a backpropagation algorithm is used to optimize the parameters. The backpropagation algorithm calculates the gradient of each parameter. Also, the number of parameters and the corresponding gradients are equal. The intermediate results, which are generated by each layer, are dominated by feature maps. The forward pass generates feature maps, and the backward pass uses them to update the parameters. Hence, feature maps need to be temporarily stored in the memory during the forward pass and before the backward pass can consume them.

The modern machine learning frameworks, e.g., TensorFlow [27] and PyTorch [28], employ a dataflow graph where the deep learning models training is modeled as a directed graph composed of a set of nodes (operations). By decomposing a deep learning model graph (discussed above) into a set of nodes or fine-grained operations (e.g., SoftMax, Sigmoid and MatMul), these frameworks greatly improve hardware utilization and system throughput. Training a deep learning model easily involves a large number of operations. In a single training step of training deep learning models, there can be tens of different operations. Each operation can be invoked hundreds of times, each of which is an operation instance.

III. METHODS

In this section, we introduce the metrics and tools we use to evaluate the performance of training deep learning models.

A. Evaluation Metrics

CPU utilization. This metric quantifies how frequently CPU is utilized during deep learning models training. This metric is defined in Equations 1 and 2.

$$CPU_Core_Utilization = \frac{T_{active}^C \times 100}{T_{total}} \% \quad (1)$$

$$CPU_Avg_Utilization = \frac{\sum_i^n (CPU_Core_Utilization^i)}{n} \quad (2)$$

Equation 1 is used to calculate the utilization of an individual CPU core. In Equation 1, T_{total} denotes the total training time; T_{active}^C indicates the active time of the CPU core. Equation 2 is used to calculate the average CPU utilization of all CPU cores. In Equation 2, n is the total number of CPU cores for training deep learning models, and i is the index of the CPU core. A larger value of $CPU_Avg_Utilization$ indicates higher CPU utilization. We want high CPU utilization to achieve high throughout processing of training operations.

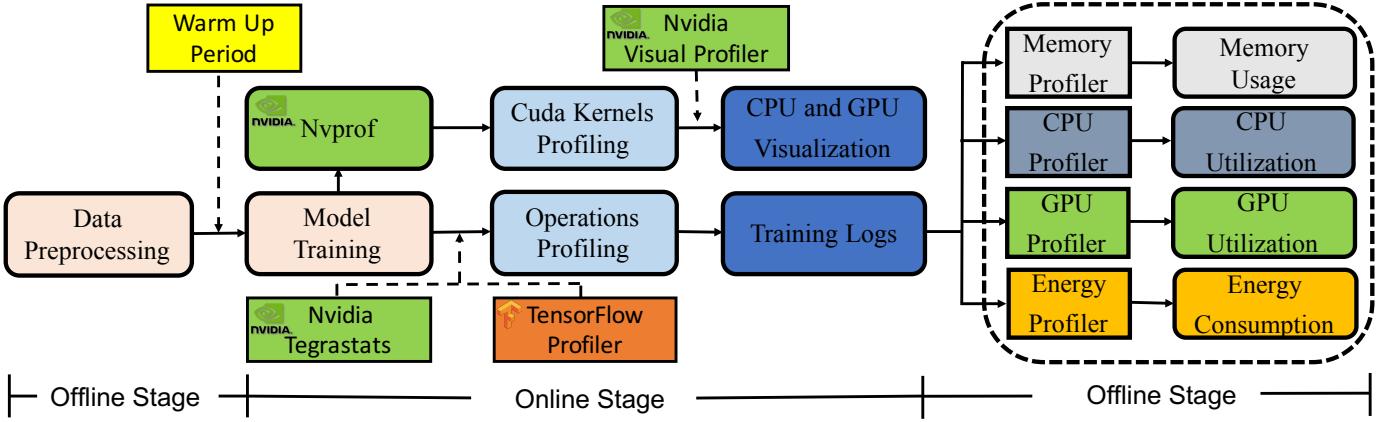


Fig. 1: Profiling tools and profiling workflow.

GPU utilization. This metric quantifies how frequently GPU is utilized during deep learning model training. This metric is defined in Equation 3.

$$GPU_Utilization = \frac{T_{active}^G \times 100}{T_{total}} \% \quad (3)$$

As shown in Equation 3, the GPU utilization is defined similar to the CPU utilization in Equation 1. We also want high GPU utilization to achieve high throughput processing of training operations.

Peak memory consumption. Training deep learning models can be memory-consuming, as a large amount of parameters, derivatives and temporary variables use the memory space. Some popular deep learning models involve a large number of parameters. For example, VGG-16 and Resnet-50 have 138 million and 25 million parameters respectively, consuming 6.3 GB and 5.8 GB memory (the batch size is 64); SqueezeNet, a small deep learning model designed for mobile devices has 5 million parameters, consuming 5.7 GB memory (the batch size is 64). The memory consumption of a deep learning model sets up a constraint on whether training the model on a mobile device is feasible. The peak memory consumption is defined as the maximum memory usage during the training process.

Energy consumption. Since a mobile device has limited battery life, reporting energy consumption of training deep learning models is critical to determine if the training is feasible within the battery life. Energy consumption is calculated based on Equation 4. During the model training, we collect power consumption of the mobile device periodically. In Equation 4, *time_interval* defines how frequently we collect power data, and *Power_Consumption_i* is the whole system power of the mobile device collected in a power sample data *i*.

$$Energy = \sum_i time_interval \times Power_Consumption_i \quad (4)$$

Throughput. This metric is used to evaluate the efficiency of the training process. Throughput in this paper is defined as how many training samples can be processed and used for training in one second. For example, when we train DenseNet40 using the batch size of 4, we can finish five training steps in one second, and each training step processes four images (samples). Hence, the throughput for training DenseNet40 is 20 samples per second.

Idle state ratio for a core. During the training, some CPU cores can be idle (i.e., the utilization is 0). Idle state ratio for a core is the percentage of the total training time that the core is idle.

B. Profiling Tools

We use the existing tools, Nvprof [29], Tegrastats [30] and TensorFlow Profiler [31], for performance analysis and characterization of training deep learning models on the NVIDIA TX2. Nvprof is a profiling tool that collects execution time of computation on CPU and GPU. Nvprof can be used to identify idle or busy states of CPU and GPU. When used for GPU, Nvprof can also be used to identify GPU kernel names (hence the names of operations running on GPU).

Tegrastats is a tool that collects hardware utilization of CPU and GPU, power consumption of hardware components (CPU, GPU, and memory) and memory consumption.

TensorFlow Profiler [31] is a tool integrated into TensorFlow runtime to perform operations statistics, including operations execution time and dependency between operations.

Nvprof and Tegrastats do not provide APIs that allow the programmer to trigger or stop profiling within the application. Nvprof and Tegrastats can run continuously as a system daemon and collect system-wide information at any moment. Simply using Nvprof and Tegrastats cannot meet the user's needs, because sometimes the user wants to correlate the profiling results (energy and memory consumption) with operations during the training process. The training process for deep learning models easily involves a large number of operations (thousands or even millions of operations in a single

time step). Collecting the profiling results for operations is challenging.

We develop a tool to address the above challenge. In particular, during the training process, we record the start and end times of all operations at each layer; We also periodically examine the execution information (including CPU and GPU utilization, power consumption of hardware components) every 10ms (we choose 10ms to control the profiling overhead). The execution information is dumped into a file with the implicit timestamp information, due to our periodical profiling method. After the training process, we associate the execution information with operations based on timing information (i.e., the start and end times of all operations). Figure 1 shows our tools and profiling workflow.

In Section IV (the section of evaluation results), the results are presented for all operations as a whole, because that allows us to easily present the results.

C. Training Deep Learning Models on NVIDIA TX2

We use the NVIDIA TX2 as our evaluation platform. This platform is an embedded system-on-module (SoM) with a dual-core NVIDIA Denver2 plus a quad-core ARM Cortex-A57 (six CPU cores in total), eight GB LPDDR4 and integrated 256-core Pascal GPU (mobile GPU). The GPU has two streaming multiprocessors (SM), and each SM has 128 cores. The eight GB memory is shared between CPU and GPU. The peak power consumption of TX2 is just 15 Watt. TX2 is a representative mobile platform. It has been commonly used in self-driving cars, robotics and drones. Many other common mobile devices, such as Snapdragon 855, Movidius Myriad2 and Nvidia Xavier, have comparable computation capability and memory capacity. Table II summarizes the major hardware features of the NVIDIA TX2.

IV. EVALUATION RESULTS

In this section, we present the evaluation results and highlight major observations.

A. Experiment Setup

Table II summarizes the deep learning models we use for evaluation. The table also lists those deep learning models that cannot be successfully trained on TX2 because of the memory constraint. Among those models, DenseNet100 and NMT can train for a few time steps, but have segmentation faults later on; VGG19, ResNet101, ResNet152 and BERT cannot get started on training at all.

We use TensorFlow v1.13 to train the deep learning models. Unless indicated otherwise, we use the default configurations for TensorFlow. Note that we use TensorFlow instead of TensorFlow Lite, although TensorFlow Lite targets on mobile devices, because of the following reasons. (1) TensorFlow Lite only supports model inference, not training. Currently, there is no training framework especially targeting on training deep learning models on mobile devices. (2) TensorFlow and TensorFlow Lite have common implementations for many

TABLE I: The Specifications of NVIDIA TX2

Hardware	Specifications
Systems	Tegra TX2 SoC
CPU1	Quad-core ARM A57 MPCore
Cache_CPU1	L1_I: 128KB, L1_D: 64KB, L2: 2MB
CPU2	Dual-core NVIDIA Denver 2 64-Bit
Cache_CPU2	L1_I: 48KB, L1_D: 32KB, L2: 2MB
GPU	NVIDIA Pascal GPU (256 CUDA Cores)
Memory	8GB 128-bit LPDDR4 Memory
Storage	32GB eMMC 5.1
Power	7.5W / 15 W

operations (e.g., convolution, matrix multiplication and max-pooling), especially those operations in the forward pass of some deep learning models.

When reporting the performance, we skip the first three training steps, because they are often used by the TensorFlow runtime system to explore hardware architectures (e.g., cache capacities and memory access latency) for performance optimization. The performance of the first three training steps is not representative of other training steps.

B. Performance Analysis

We study the training performance from the following perspectives: hardware (CPU and GPU) utilization, power consumption, and peak memory consumption.

Hardware Utilization

Figure 3 shows the CPU and GPU utilization when we train the deep learning model Inception V1. The figure shows the hardware utilization for three training steps. Since the NVIDIA TX2 includes 6 CPU cores, we use six subgraphs to show the utilization of each of the six cores: the first two subgraphs show the utilization of two Denver2 cores, and the rest of them shows the utilization of four A57 cores. We have the following observations.

Observation 1: The GPU utilization is generally much higher than the CPU utilization. In most of the times, the GPU utilization is close to 100%, while each CPU core utilization ranges from 0% to 60%. Also, when the GPU utilization is high, the CPU utilization tends to be low, and vice versa. This indicates that the workload is not balanced well between CPU and GPU. There seems a lack of effective coordination between CPU and GPU. This observation is general and exists in many deep learning models (e.g., Inception V1, DCGAN, Resnet50, Xception).

Our further investigation reveals that when the GPU utilization is low, CPU is either busy with data fetching from storage (SSD) to main memory, or working on small operations that are not worth to be offloaded to GPU due to the large data copy overhead; When the GPU utilization is high, CPU is working on a few small operations, and most of CPU cycles are wasted.

Such an observation also exists in servers, but the difference is that the utilization difference between CPU and GPU on servers tends to be larger [49], because GPU on servers are much more powerful than CPU on servers and hence more operations (after kernel fusing) tend to be scheduled on GPU.

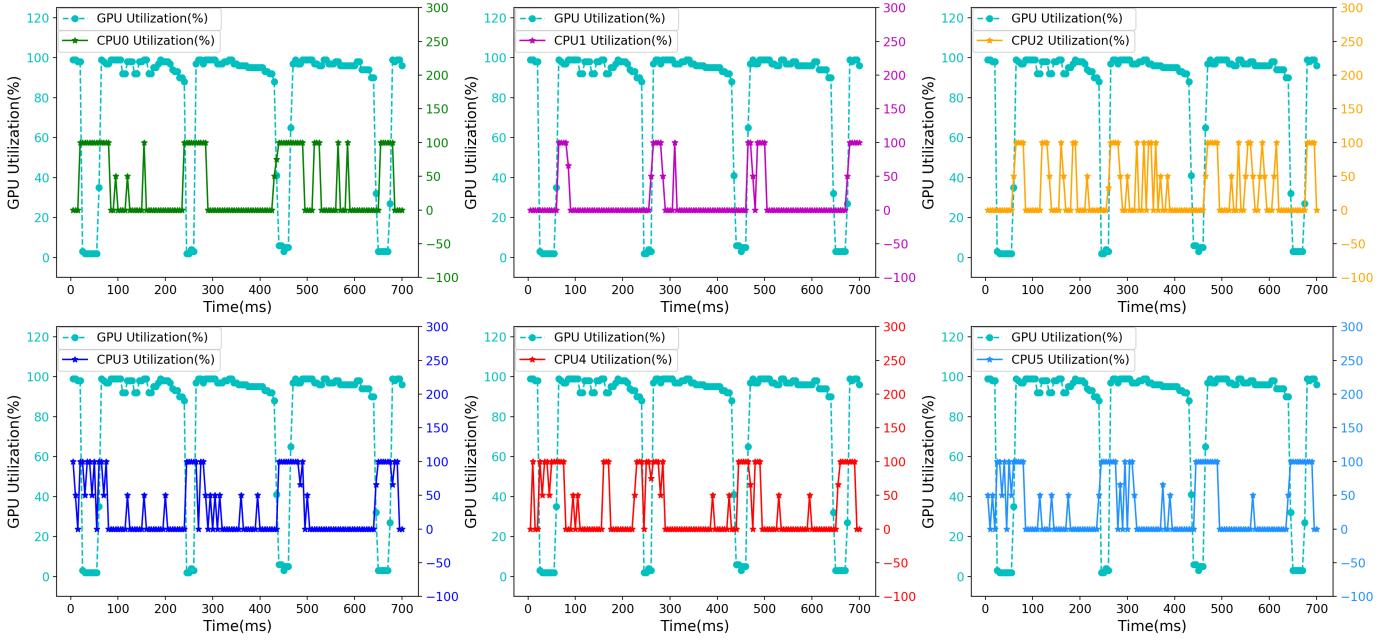


Fig. 2: Utilization of GPU and six CPU cores for training Inception V1.

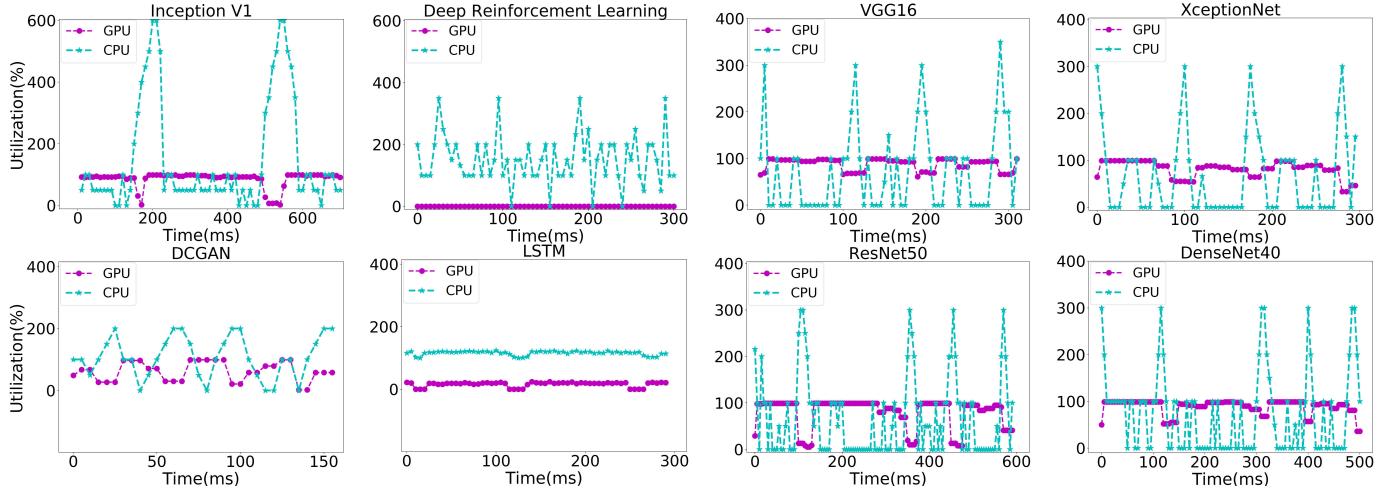


Fig. 3: CPU and GPU utilization of different models.

Observation 2: The utilization of GPU and each core in CPU is predictable. The utilization shows a periodical pattern where busy cycles alternate with less busy cycles. A period of the pattern corresponds to one time step. Across time steps, such a pattern repeatedly appears. This indicates that the computation across time steps remains stable and hence is highly predictable. This observation is consistent with the existing work that leverages predictability of deep learning workloads for performance optimization [50], [51].

Such an observation also exists in servers. Since this observation is determined by the process of training deep learning models that repeatedly goes through a computation graph (and not hardware architecture-related), this observation is general and independent of hardware architectures.

Observation 3: The GPU utilization is sensitive to the batch size, while the CPU utilization is not. Figure 4 shows the CPU and GPU utilization when the batch size changes. For DenseNet, the GPU utilization increases from 81.6% to 96.4% as the batch size changes from 4 to 64. For SqueezeNet and ResNet50, the GPU utilization increase from 71.4% to 84.5% and from 85.6% to 95.6% respectively, as we increase the batch size. However, for the CPU utilization, there is only 2.1% difference on average across models.

As we increase the batch size, the memory footprint increases and computation for operations also increases. Since GPU works on most computation-intensive operations during the training, its utilization also increases as more computation requires more thread-level parallelism. The CPU utilization

TABLE II: Descriptions for deep learning models in our evaluation

Model	#Layers	Dominant layer	#Flops	#Parameters	Training dataset	Batch size	Successful training?	Application domain
DenseNet40_12 [32]	40	CONV	30M	1M	Cifar-10 [33]	1-64	✓	Computer Vision
ResNet50 [34]	50	CONV	4G	98M	Cifar-10	1-64	✓	Computer Vision
SqueezeNet [35]	40	CONV	837M	5M	Cifar-10	1-64	✓	Computer Vision
VGG16 [36]	16	CONV	4G	134M	Cifar-10	1-64	✓	Computer Vision
XceptionNet [37]	39	CONV	N/A	23M	Cifar-10	1-64	✓	Computer Vision
InceptionV1 [38]	22	CONV	30M	5M	Cifar-10	1-64	✓	Computer Vision
Char-CNN [39]	2	LSTM+CONV	23M	1M	Shakespeare [40]	1-64	✓	Natural Language Processing
DCGAN [41]	40	CONV	30M	1M	Cifar-10	1-64	✓	Image Generation
Deep RL [42]	4	CONV	N/A	N/A	Atari 2600 games [43]	1-64	✓	Robotics Control
AlexNet [44]	8	CONV	727M	60M	Cifar-10	1-64	✓	Computer Vision
VGG19 [36]	19	CONV	20G	548M	Cifar-10	1-64	x	Computer Vision
BERT [45]	12	Embedding	N/A	110M	SQuAD [46]	1-64	x	Natural Language Processing
ResNet101 [34]	101	CONV	8G	155M	Cifar-10	1-64	x	Computer Vision
ResNet152 [34]	152	CONV	11G	220M	Cifar-10	1-64	x	Computer Vision
DenseNet100 [32]	100	CONV	31M	7M	Cifar-10	1-64	x	Computer Vision
seq2seq [47]	2	LSTM	28G	348M	IWSLT15 [48]	1-64	x	Natural Language Processing

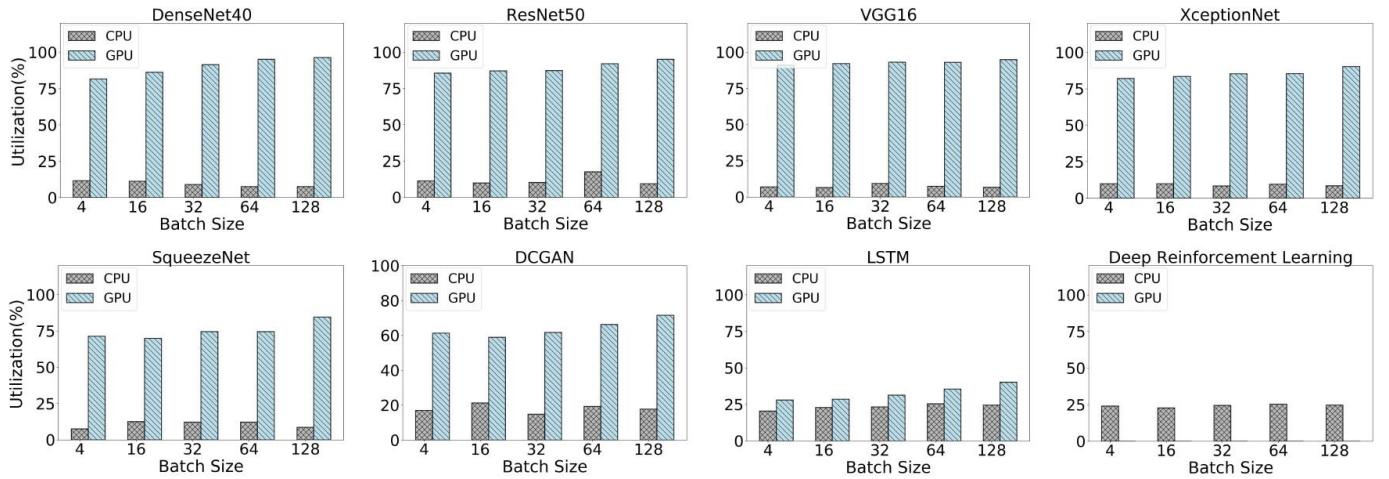


Fig. 4: CPU and GPU utilization of different models.

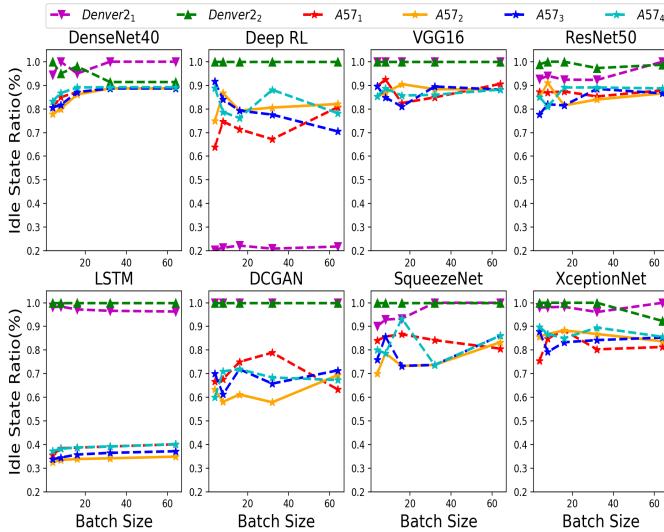


Fig. 5: Idle state ratio of six CPU cores for different models.

does not increase very much, because CPU works on small operations and most of data objects in those operations can

be in caches. Slight increase of memory footprint due to the increase of the batch size does not cause extra cache misses and does not significantly impact execution time.

Such an observation also exists in servers. But the variance of GPU utilization on servers does not change as much as that on mobile devices, because GPU on servers have more cores and hence offers more thread-level parallelism to work on increased computation as we increase the batch size. [51]

Observation 4: Different cores have different utilization during the training. TX2 has six heterogeneous cores: Two of them are Denver2 and four of them are A57. As Figure 5 shows, the utilization of each core changes differently, as the batch size increases. There is no obvious correlation between the changes of the utilization across cores. We also notice that the two Denver2 cores have the highest idle state ratio (as high as 65%) among all CPU cores, which indicates a large room for performance improvement.

We do not have the above observation on servers, because servers (especially x86 servers) usually do not have heterogeneous CPU cores [52].

Figure 4 shows the GPU utilization of different deep learning models from different application domains. The models

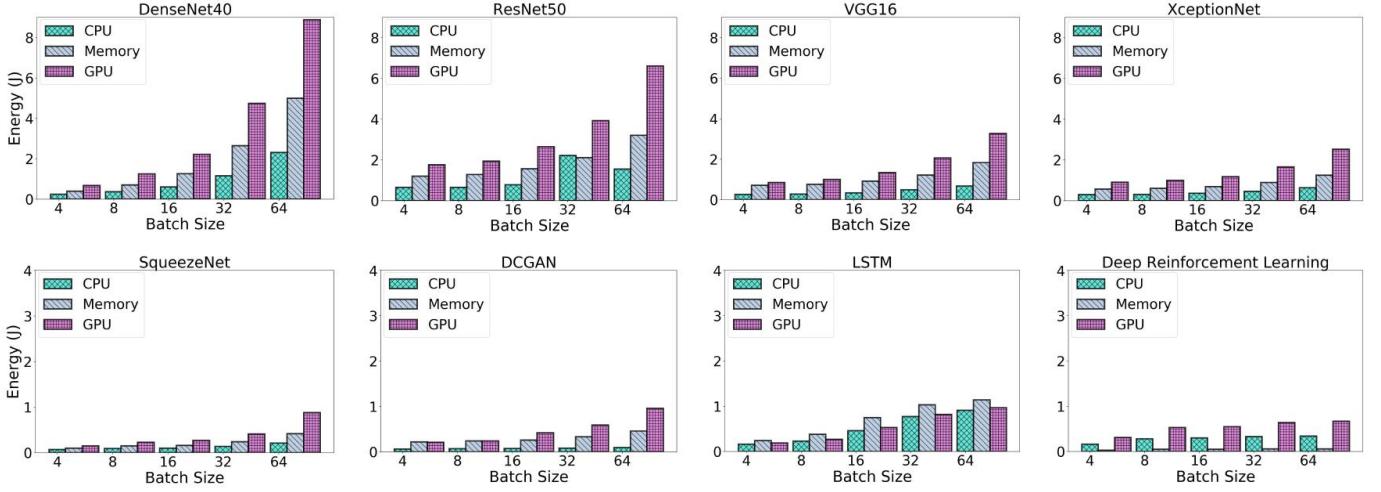


Fig. 6: Energy consumption of different models.

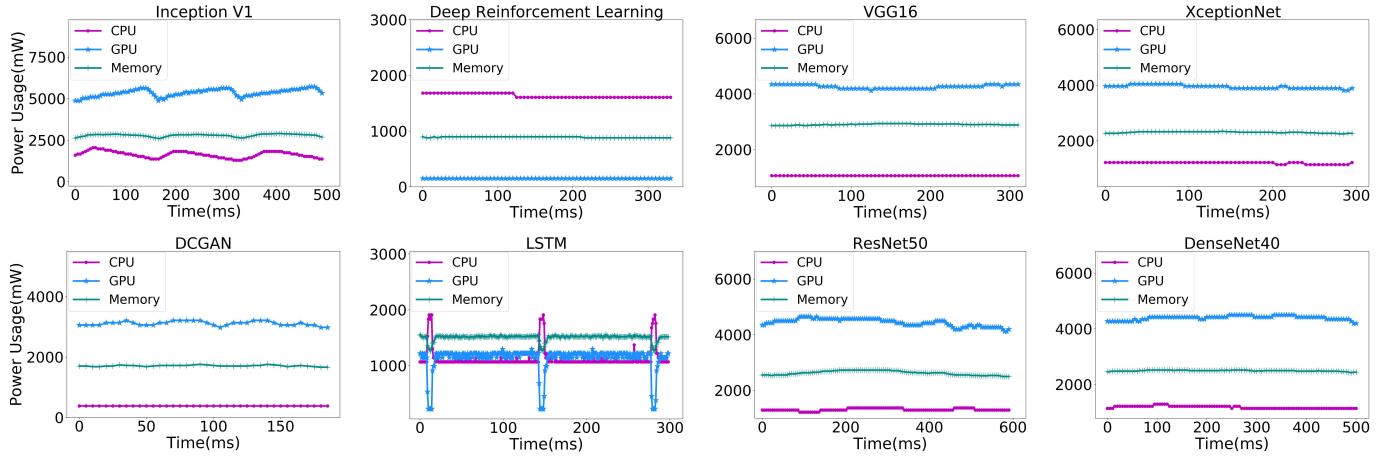


Fig. 7: Power usage of different models in three iterations.

from the domain of computer vision have the similar GPU utilization, hence we show Inception V1 as a representative of this domain. We choose other models including LSTM and DCGAN to represent different application domains.

Observation 5: The GPU utilization varies on different application domains. In Figure 4, we find the LSTM model (the domain of natural language processing) has a low GPU utilization (only about 25%), while the models from the domain of computer vision (e.g., ResNet) have higher GPU utilization (95%). Those models from computer vision has high GPU utilization, because they often employ convolution which is easy to leverage SIMD (Single Instruction Multiple Thread) on GPU and reach high GPU utilization. In LSTM, operations often have dependency and there is lack of available thread-level parallelism. The observation is consistent with the existing work [53]–[55] that LSTM has lower utilization than computer vision models.

Such an observation also exists in servers. Since the GPU utilization is heavily impacted by the application domain,

Observation 5 is general and independent of hardware architectures [15].

Power and Energy Consumption

Figures 7 and 6 show power and energy consumption of GPU, CPU, and memory. Figure 7 shows how power consumption changes for three training steps. Figure 6 shows energy consumption for one training step, when the batch size changes. Energy consumption is calculated based on Equation 4 with the time interval of 5ms.

Observation 6: GPU is a power-consuming hardware component, but for some deep learning model, the memory consumes more power than GPU. Figure 4 shows that for the domain of computer vision, GPU is the most time-consuming hardware component when we train deep learning models (especially CNN models) such as ResNet50 and VGG16. In those models, GPU consumes 4× and 2× of power consumption of CPU and memory respectively. GPU consumption can take up to 57.4% of the whole system power. Different from the above examples, the memory is the most power-

consuming hardware component (not GPU), when we train LSTM. Compared with the CNN models, LSTM has relatively bad data locality and causes more intensive memory accesses. As a result, the memory, shared between GPU and CPU, draws large power consumption.

Such an observation does not exist in servers. In servers, GPU (including its global memory) is the most power-consuming (e.g., NVIDIA V100 takes up to 250 Watt (more than half of the system power) when training LSTM, while the memory (main memory) takes only 20%-30% of the total system power.)

Observation 7: The power consumption across training steps is predictable. Similar to the hardware utilization, the power consumption of hardware components shows a periodical pattern. This pattern is highly predictable across training steps. Figure 7 shows such results. On servers, we have the similar observations.

Observation 8: As we increase the batch size, the energy consumption increases as well, but not in a proportional way. Figure 6 shows the results to support this observation. As we increase the batch size from 4 to 64 (16x increase), the energy consumption of the whole system increases as well. However, the increase of the energy consumption is at least 2.2x and at most 10.5x, less than 16x when we change the batch size.

Also, different models show quite different energy consumption. Among the five deep learning models for computer vision, DenseNet40 is the most energy-consuming one, while the SqueezeNet is the most energy efficient one. The above conclusion is true as we run the training to completion (including all time steps). The above observation also exists in servers.

Consistent with the results of power consumption, we notice that for some models (e.g., DenseNet40), GPU is the most energy-consuming one, while for LSTM, the memory is the most energy-consuming one.

Peak Memory Consumption

The memory is one of the key limiters for deep learning models training on mobile devices. Some large models, e.g., ResNet101 and VGG19, consume more than 10 GB memory for training, while TX2 only has 8 GB memory. Those models cannot be trained on TX2. In our study, we aim to study the impact of the batch size on the memory consumption of deep learning models. Different from on servers, on mobile devices we must carefully choose the batch size, not only for good training accuracy as on servers, but also for acceptable memory consumption.

For training (especially CNN and RNN), the memory is consumed by the following critical variables: parameters (including weights and bias), gradients, input data, and intermediate data. Among them, the intermediate data is the most memory consuming. The intermediate data includes the work space and feature map. The work space is the memory consumed by the machine learning framework (e.g., TensorFlow or PyTorch). The memory consumption of the work space varies for different frameworks. The feature map, sitting in the middle of two

neighbor layers of a CNN or RNN model, is generated by one layer, and used as input of the next layer.

Observation 9: Choosing a good batch size is critical to be able to train deep learning models on mobile devices. Figure 9 shows memory usage as we change the batch size. As expected, parameters, input data and gradients remain constant, as we increase the batch size. But the memory consumption of intermediate data increases significantly, as we increase the batch size. For example, for DenseNet40, when the batch size increases from 4 to 64, the memory consumption of intermediate data increases from 2.2 GB to 5.9 GB. When we use larger batch sizes (12nd 256), we run out of memory for all models.

Throughput

To quantify throughput, we employ a common metric, training samples per second, instead of using images per iteration (training step) as in some deep learning models, because of the following two reasons. First, our collection of deep learning models includes CNN, RNN and Deep reinforcement learning models, which means that the training samples for some models are not images. For example, the training samples are sentences for some RNNs (e.g., seq2seq). Second, as we change the batch size for evaluation, the number of training samples for a training step changes as well, which indicates that the execution time per training step (iteration) changes. Hence, using “second” (the time metric) instead of “iteration” makes more sense.

Observation 10: Throughput increases as the batch size increases. Figure 8 shows the throughput as we change the batch size. For all models, the throughput increases as the batch size increases. For example, for ResNet50, the throughput increases from 9 to 55 samples per second as the batch size increases from 4 to 64.

The above observation can also be seen in servers [15].

Observation 11: Across models, throughput changes differently, as we increase the batch size. In Figure 8, the throughput of the deep reinforcement learning model increases from 889 to 13,618 samples per second as the batch size increases from 4 to 64 (15.3x speedup). However, for DenseNet and ResNet50, such throughput speedup is 1.7x and 6.1x, respectively. The deep reinforcement learning model has big throughput speedup as we increase the batch size. This is because the training time of the deep reinforcement learning does not change too much, as we increase the batch size. As a result, the throughput increases significantly, as we increase the batch size.

The above observation also exists on servers [15].

Study on modeling accuracy

Training a deep learning model on a mobile device is different from that on a server, because training samples can be dynamically generated when the mobile device is used. For example, training DenseNet40 can be based on training samples (images) collected at the user’s day time. In this evaluation, we evaluate a scenario where the user uses a mobile device to generate 64 images per day, and those images are used to train a deep learning model (DenseNet40). We also

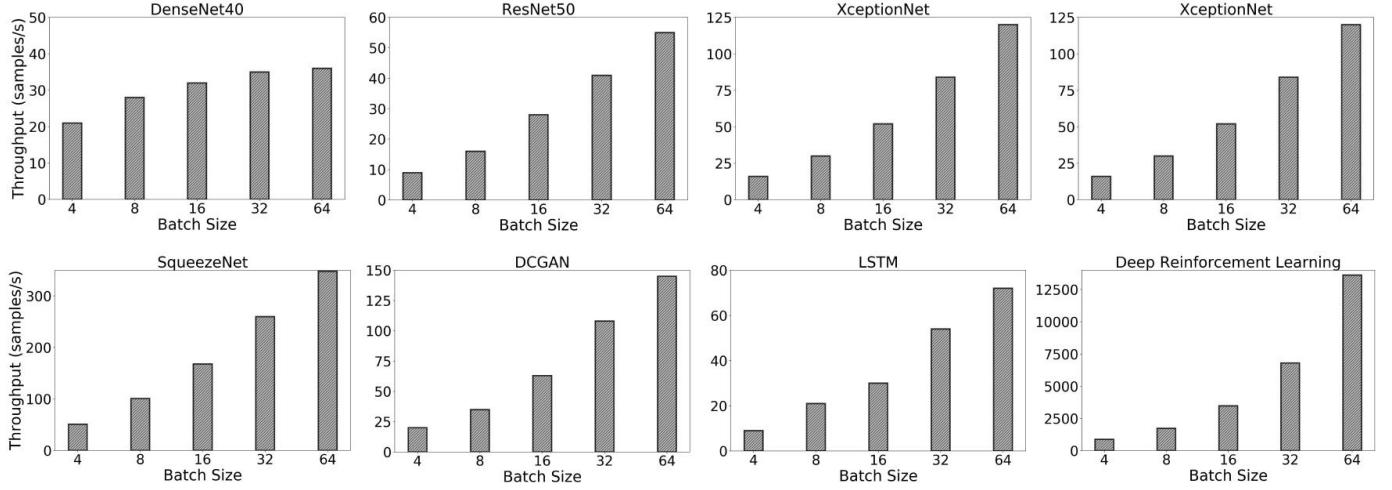


Fig. 8: Throughput of deep learning models.

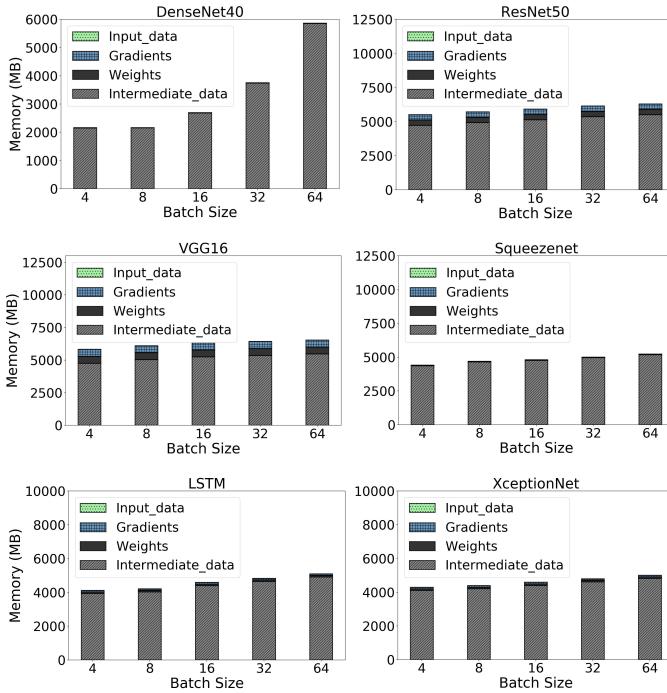


Fig. 9: Memory usage of deep learning models.

assume that the model, before started to be trained, is already trained on a server, but needs to be trained further, using the user's new training samples. Figure 10 shows the variance of the accuracy, as we use the above training method for 17 days. As the day 0, the training accuracy is 60.65%, because the model is already trained on a server.

Observation 12: Training a deep learning model on a mobile device can slowly increase training accuracy. Figure 10 reveals that the accuracy of DenseNet40 increases from 60.65% to 61.32% (using three different batch sizes). The above observation reveals that using the above method

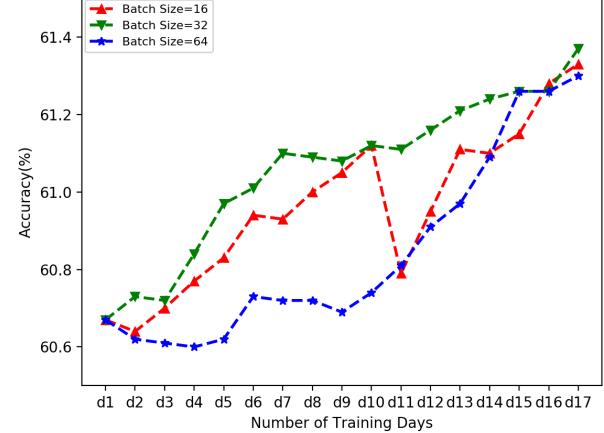


Fig. 10: The accuracy variance as we increase training samples at a daily base to train DenseNet40.

does slowly increases the accuracy. In this special scenario, depending on whether the user has high requirement on the model accuracy, the training on the mobile device can continue as more training samples are collected or stop.

V. DISCUSSION AND FUTURE RESEARCH DIRECTIONS

Feasibility of training deep learning models on mobile devices. Our work demonstrates the feasibility of training some deep learning models on a mobile device. Most of the models we study are traditionally trained on a server, and seldom trained on any mobile device. Those deep learning models come from various application domains, and have potential to provide new services for mobile users.

Our observation 9 reveals that choosing an appropriate batch size has a big impact on whether training a deep learning model is feasible. Furthermore, it is well known that the batch size has an impact on the model accuracy. Hence, there is a non-trivial tradeoff between the training feasibility and

accuracy on mobile devices. Such a tradeoff deserves further study.

Hardware utilization. Mobile devices often offer rich hardware heterogeneity (e.g., there are two types of CPU cores in the NVIDIA TX2), richer than x86 servers. However, such hardware heterogeneity is not leveraged well in the current machine learning frameworks. This fact is pronounced by two observations: (1) The utilization of all CPU cores is relatively low (comparing with GPU); (2) The heterogeneity of CPU cores is completely ignored. As a result of such fact, the CPU cycles are wasted and the computation power of specialized CPU cores (e.g., ARM Cortex-A57) is not fully utilized.

The recent work from Facebook [49] reveals that the performance difference between CPU and GPU on mobile devices is smaller than that on servers. Based on this work and our observations, we see great opportunities to improve the current scheduling mechanism in the machine learning frameworks. Only using GPU for computation-intensive operations may not be a good scheduling strategy. Instead, balancing workloads on CPU and GPU to maximize system throughput (for finishing operations) is a better one.

Energy consumption. Mobile devices are more sensitive to energy consumption than servers. Training deep learning models on mobile devices raises concerns on whether the battery life is good enough to support training. Although the recent work suggests to train deep learning models when the mobile device is charged [16], [56], [57], the charging time can be longer than the training time. The batter may still be needed to finish training. Hence, minimizing energy consumption during training is critical. Our observation reveals that the memory can be more energy-consuming than CPU and GPU, when we train some deep learning networks. Reducing energy consumption of the memory is necessary for mobile devices. How to reduce energy consumption of the memory without impacting performance (execution time) is an open topic.

Predictability of workload characterization. The workload of training deep learning networks is predictable, which means execution time, hardware utilization and power consumption show a repetitive pattern across training steps. Such predictability allows us to apply dynamic profiling on a few training steps to collect workload characterization, based on which we guide operation scheduling and power management in the future training steps. Predictability of execution time during the training has been leveraged in the existing work [50], [51]. We expect to leverage the predictability of other characterization in the future work.

VI. RELATED WORK

Performance optimization of deep learning model training. Some recent works [16]–[26], [58] have demonstrated the promise of training neural networks (NN) on mobile devices. They are focused on exploring performance optimization in the perspectives of algorithm and system. For example, Mao et al. [58] implement a distributed mobile learning system that trains a neural network by multiple devices of the same local network

in parallel. They design a scheduler to adapt the training configuration for heterogeneous mobile resources and network circumstances. Bonawitz et al. [16] develop a federated learning system to achieve NNs training on mobile platforms using TensorFlow. Some practical issues have been addressed, e.g., local data distribution, unreliable device connectivity and limited on-board resources. Konečný et al. [19] use parameter compression techniques to reduce the uplink communication costs in federated learning. This paper is orthogonal to the above works. Our comprehensive model profiling and analysis can be used to develop more efficient NN training schemes on mobile devices.

Zhu et al. [15] study the training performance and resource utilization of eight deep learning model models implemented on three machine learning frameworks running on servers (not mobile devices) across different hardware configurations. However, they do not consider power and energy efficiency. In contrast, our work is focused on deep learning models training on mobile devices.

Profiling of deep neural network inference. Many works have been conducted to analyze the performance and resource utilization of machine learning workloads (inference, not training) on mobile devices [4]–[10]. Lu et al. [4] measure the performance and resource usage of each layer in CNNs running on mobile CPUs and GPUs. Based on the results of profiling and modeling, they implement a modeling tool to estimate the compute time and resource usage of CNNs. However, they only consider CNNs, but not RNNs or reinforcement learning models which are also important for mobile applications. Hanhirova et al. [5] profile the performance of multiple CNN-based models for object recognition and detection on both embedded mobile processors and high-performance server processors. They find that there exists significant latencythroughput trade-offs. Unfortunately, the above works only study the inference of CNNs. On the contrary, we profile and analyze the performance and resource requirements of CNNs, RNNs and deep reinforcement learning models training on mobile devices.

VII. CONCLUSIONS

Training deep learning networks on mobile devices is emerging because of increasing computation power on mobile hardware and the advantages of enabling high user experiences. The performance characterization of training deep learning models on mobile devices is largely unexplored, although understanding the performance characterization is critical for designing and implementing deep learning models on mobile devices. This paper is the first work that comprehensively studies the performance of training deep learning network on a mobile device. Our study is based on a set of profiling tools on mobile devices, and uses a set of representative deep learning models from multiple application domains. We reveal many research opportunities as a result of our study. We hope that our study can motivate future study on optimizing performance of training deep learning networks on mobile devices.

REFERENCES

- [1] N. D. Lane, P. Georgiev, and L. Qendro, “Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning,” in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 283–294, ACM, 2015.
- [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*, pp. 68–81, ACM, 2017.
- [3] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, “Deepcache: Principled cache for mobile deep vision,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 129–144, ACM, 2018.
- [4] Z. Lu, S. Rallapalli, K. Chan, and T. La Porta, “Modeling the resource requirements of convolutional neural networks on mobile devices,” in *Proceedings of the 25th ACM international conference on Multimedia*, pp. 1663–1671, ACM, 2017.
- [5] J. Hanhirova, T. Kämäriäinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, “Latency and throughput characterization of convolutional neural networks for mobile computer vision,” in *Proceedings of the 9th ACM Multimedia Systems Conference*, pp. 204–215, ACM, 2018.
- [6] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, “Fathom: Reference workloads for modern deep learning methods,” in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 1–10, IEEE, 2016.
- [7] S. Shi, Q. Wang, P. Xu, and X. Chu, “Benchmarking state-of-the-art deep learning software tools,” in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pp. 99–104, IEEE, 2016.
- [8] “cnn-benchmarks.” <https://github.com/jcjohanson/cnn-benchmarks>.
- [9] “convnet-benchmark.” <https://github.com/soumith/convnet-benchmarks>.
- [10] “DeepBench.” <https://github.com/baidu-research/DeepBench>.
- [11] “Apple A12.” https://en.wikipedia.org/wiki/Apple_A12.
- [12] “Snapdragon 855.” <https://www.qualcomm.com/products/snapdragon-855-mobile-platform>.
- [13] “Kirin 980.” <https://en.wikichip.org/wiki/hisilicon/kirin/980>.
- [14] “Nvidia Jetson Xavier.” <https://developer.nvidia.com/embedded/buy/jetson-agx-xavier-devkit>.
- [15] H. Zhu, M. Akroot, B. Zheng, A. Pelegris, A. Jayaraman, A. Phanishayee, B. Schroeder, and G. Pekhimenko, “Benchmarking and analyzing deep neural network training,” in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 88–100, IEEE, 2018.
- [16] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingberman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, et al., “Towards federated learning at scale: System design,” *arXiv preprint arXiv:1902.01046*, 2019.
- [17] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, “Federated multi-task learning,” in *Advances in Neural Information Processing Systems*, pp. 4424–4434, 2017.
- [18] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, “Applied federated learning: Improving google keyboard query suggestions,” *arXiv preprint arXiv:1812.02903*, 2018.
- [19] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.
- [20] H. Zhu and Y. Jin, “Multi-objective evolutionary federated learning,” *arXiv preprint arXiv:1812.07478*, 2018.
- [21] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data,” *arXiv preprint arXiv:1806.00582*, 2018.
- [22] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, “Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data,” *arXiv preprint arXiv:1811.11479*, 2018.
- [23] X. Qi and C. Liu, “Enabling deep learning on iot edge: Approaches and evaluation,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 367–372, IEEE, 2018.
- [24] W. Du, X. Zeng, M. Yan, and M. Zhang, “Efficient federated learning via variational dropout,” 2018.
- [25] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, “On the convergence of federated optimization in heterogeneous networks,” *arXiv preprint arXiv:1812.06127*, 2018.
- [26] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, “Adaptive federated learning in resource constrained edge computing systems,” *learning*, vol. 8, p. 9, 2018.
- [27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., “Tensorflow: a System for Large-scale Machine Learning,” in *USENIX Symposium on Operating Systems Design and Implementation*, 2016.
- [28] Adam Paszke and Sam Gross and Soumith Chintala and Gregory Chanan, “PyTorch.” <https://pytorch.org/>.
- [29] “Nvidia Nvprof.” <https://docs.nvidia.com/cuda/profiler-users-guide/index.html>.
- [30] “Nvidia Tegrastats.” <https://docs.nvidia.com/jetson/l4t/index.html>.
- [31] “TensorFlow Profiler.” https://www.tensorflow.org/api_docs.
- [32] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.
- [33] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” tech. rep., Citeseer, 2009.
- [34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [35] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and $\frac{1}{10}$ mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.
- [36] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [37] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.
- [38] 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*, pp. 1–9, 2015.
- [39] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in *Advances in neural information processing systems*, pp. 649–657, 2015.
- [40] A. Singh and X. J. Zhu, eds., *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20–22 April 2017, Fort Lauderdale, FL, USA*, vol. 54 of *Proceedings of Machine Learning Research*, PMLR, 2017.
- [41] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [42] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [43] M. G. Bellemare, J. Veness, and M. Bowling, “Investigating contingency awareness using atari 2600 games,” in *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
- [44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [46] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “Squad: 100,000+ questions for machine comprehension of text,” *arXiv preprint arXiv:1606.05250*, 2016.
- [47] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, pp. 3104–3112, 2014.
- [48] “IWSLT Evaluation 2015,” <https://sites.google.com/site/iwsltEvaluation2015/>.
- [49] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. M. Hazelwood, E. Isaac, Y. Jia, B. Jia, et al., “Machine learning at facebook: Understanding inference at the edge,” in *HPCA*, pp. 331–344, 2019.
- [50] M. Sivathanu, T. Chugh, S. S. Singapuram, and L. Zhou, “Astra: Exploiting Predictability to Optimize Deep Learning,” in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

- [51] J. Liu, D. Li, G. Kestor, and J. Vetter, “Runtime Concurrency Control and Operation Scheduling for High Performance Neural Network Training,” in *International Symposium on Parallel and Distributed Systems*, 2019.
- [52] S. Mittal and J. S. Vetter, “A survey of cpu-gpu heterogeneous computing techniques,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 69, 2015.
- [53] M. Zhang, S. Rajbhandari, W. Wang, and Y. He, “Deepcpu: Serving rnn-based deep learning models 10x faster,” in *2018 {USENIX} Annual Technical Conference ({USENIX} {ATC} 18)*, pp. 951–965, 2018.
- [54] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, “Processing-in-memory for energy-efficient neural network training: A heterogeneous approach,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 655–668, IEEE, 2018.
- [55] Z. Jia, M. Zaharia, and A. Aiken, “Beyond data and model parallelism for deep neural networks,” *arXiv preprint arXiv:1807.05358*, 2018.
- [56] M. Mohri, G. Sivek, and A. T. Suresh, “Agnostic federated learning,” *CoRR*, vol. abs/1902.00146, 2019.
- [57] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, *et al.*, “Communication-efficient learning of deep networks from decentralized data,” *arXiv preprint arXiv:1602.05629*, 2016.
- [58] J. Mao, Z. Qin, Z. Xu, K. W. Nixon, X. Chen, H. Li, and Y. Chen, “Adalearn: An adaptive distributed mobile learning system for neural networks,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 291–296, IEEE, 2017.