Performance Prediction of Deep Learning Applications Training in GPU as a Service Systems

Marco Lattuada · Eugenio Gianniti · Danilo Ardagna · Li Zhang

Received: date / Accepted: date

Abstract Data analysts predict that the GPU as a Service (GPUaaS) market will grow from US$700 million in 2019 to $7 billion in 2025 with a compound annual growth rate of over 38% to support 3D models, animated video processing, and gaming. GPUaaS adoption will be also boosted by the use of graphics processing units (GPUs) to support Deep learning (DL) model training. Indeed, nowadays, the main cloud providers already offer in their catalogs GPU-based virtual machines pre-installed with the popular DL framework (like Torch, PyTorch, TensorFlow, and Caffe) simplifying DL model programming operations.

Motivated by these considerations, this paper studies GPU-deployed neural networks (NNs) and tackles the issue of performance prediction, particularly with respect to NN training times. The proposed approach is based on machine learning and exploits two main sets of features which describe, on one hand, the network architecture and the hyper-parameters, on the other, the hardware characteristics of the target deployment. Such data enable the learning of multiple linear regression models, which, coupled with an established feature selection technique, become accurate prediction tools, with errors below 11 % on average.

An extensive experimental campaign, performed both on public and in-house private cloud deployments, considers popular deep NNs used for image classification and speech transcription and shows that prediction errors remain small even when extrapolating outside the range spanned by the input data. This has important implications for the models' applicability: in this way, it is possible to investigate the impact on the performance of different GPUaaS deployment or hardware upgrades even without conducting an empirical investigation on the specific target device or to evaluate the changes in training time when the number of inner modules in the deep neural networks varies.

M. Lattuada, E. Gianniti, D. Ardagna
Politecnico di Milano
E-mail: name.lastname@polimi.it

L. Zhang
Amazon E-mail: lzhangza@amazon.com

1 Introduction

As the amount of data to process increases, the main purpose of the GPU has changed. Initially, the GPU was developed for high-performance graphics work. Since 2000, GPUs are widely used to support high-performance data processing and simulations and there are several studies dealing with GPU adoption in cloud computing Jun et al. (2018). GPU as a Service (GPUaaS) is a recent novel option offered by cloud providers among their infrastructural services to access GPUs on an on-demand pay-per-use basis. According to data analysts, GPUaaS market's value will grow from US$700 million in 2019 to $7 billion in 2025 with a compound annual growth rate of over 38 % Global Market Insights (2019) to support 3D models, animated video processing, gaming, and deep learning (DL).

In recent years, with the increasing attention to artificial intelligence, studies on DL have indeed become more and more popular. DL workloads usually need to use GPUs to accelerate their execution since, e.g., model training can obtain from 5 up to 40× time improvement when compared to CPU deployments Bahrampour et al. (2015); Madougou et al. (2016).

Nowadays, DL methods are fruitfully exploited in a wide gamut of products across industries, ranging from medical diagnosis to public security. Among them, neural networks (NNs) are the most popular technique, whose different variants aim at solving different classes of problems. For example, convolutional neural networks (CNNs) were successfully adopted for image recognition and classification tasks Krizhevsky et al. (2012). More complex architectures, such as recurrent neural networks (RNNs), have also been proposed to cope with the time dimension. RNNs exhibit a dynamic behavior and are widely used today for speech recognition or machine translation Khomenko et al. (2017). At the same time, with the emergence of more and more deep learning frameworks such as Torch Torch (2018), PyTorch PyTorch (2018), TensorFlow TensorFlow (2018), and Caffe Caffe2 (2018) the use of GPUs is no longer a cumbersome task, making more and more people begin to write GPU-based programs.

The main cloud players are offering today GPU-based virtual machines (VMs) where DL frameworks are already available Amazon (2018a); Google (2018b); Microsoft (2018). The use of cloud platforms is beneficial for end-users since they can access optimized instances on demand, thus distributing their workloads across different numbers and types of GPUs. In some cases, advanced services (like Amazon Rekognition Amazon (2018b) or Google AutoML Google (2018a)) allow end-users to easily add image and video analyses to final applications starting only from labeled images, without the need to even define a model and without any DL expertise. For these reasons, the public delivery model is expected to dominate the market in 2024. However, consumers will likely adopt private and hybrid clusters too, due to their security features Global Market Insights (2019).

In spite of the widespread adoption of DL systems, still there are few studies investigating how, for instance, the training time changes when running on different GPUs or by varying the number of training iterations or the batch size Bahrampour et al. (2015); Hadjis et al. (2016). DL applications are characterized by a large number of design choices that often do not apply readily to other domains or hardware configurations, up to the point that even advanced users with considerable DL expertise fail at identifying the best performing configuration settings Hadjis et al. (2016). On one side, performance modeling can help cloud providers to establish service level agreements with end-users. On the other side, performance modeling can allow cloud end-users to predict the budget to train or run production DL models in the cloud.

This paper proposes an approach to learn performance models for NNs running on multiple and heterogeneous GPU systems. The main metric under investigation is the NN training time, but the approach can be easily extended to estimate the forward time for a DL model deployed in production Gianniti et al. (2018b).

The final goal is to lead cloud users to predict: (i) the training time as a function of the number of iterations and batch size, (ii) how the training time would change by selecting different types or number of GPUs, and (iii) what would be the impact on the training time of adopting more complex networks based on a fixed structure. This last aspect is of paramount importance, since modern DL networks (e.g., VGG Simonyan and Zisserman (2014), ResNet He et al. (2015), and Inception Szegedy et al. (2015)) have the same basic structure. The number of inner modules (which are usually replicated to improve the DL model's accuracy) is adjusted to fine-tune training and cross-validation errors, so as to obtain production quality models.

In particular, this paper proposes machine learning (ML) models based on linear regression, able to learn a performance model from a collection of experimental runs of the target DL networks. Three popular CNNs for image classification and one RNN targeting speech recognition, for which the training on multiple GPUs is particularly challenging from the performance modeling perspective, have been considered. Moreover, multiple frameworks (TensorFlow TensorFlow (2018), the most widely used, and PyTorch, the fastest growing in the NNs community den Bakker (2017)) and heterogeneous GPU environments, including both multiple VM instance types offered by a public cloud provider and in-house servers representative of private cloud deployments, have been considered.

To improve the performance model's generalization capabilities (for instance, to predict the performance of a particular GPU model when the profiling data used for learning is collected on different GPUaaS deployment or in-house hardware), feature engineering and selection are performed by relying on the stepwise method, widely used in statistics Draper and Smith (1966).

As empirical analyses will demonstrate, small scale profiling experiments are enough to obtain accurate performance models, which are able to extrapolate the training time estimation with a batch size of different scale, or with more powerful hardware, mitigating the profiling cost of this approach.

In the experimental campaign, the four NN models have been run on six different VM instance types on Microsoft Azure and on two different in-house servers. Overall, four different NVIDIA GPU models widely used at the time of writing have been analyzed. An extended analysis of the most important features to be included in the ML models is also provided.

Our results show that performance models are accurate enough to predict the training time under a number of scenarios of practical interest and allow users to estimate, e.g., how training time changes by changing GPU type or number, or changing the network's inner modules number, overall yielding relative errors about 27 % in the worst case.

This paper is organized as follows. Section 2 reviews related work. Section 3 describes what are the inputs of the proposed performance models and how they are built. Section 4 presents the experimental setup adopted for training the performance models, whose validation is presented in Section 5. Conclusions are finally drawn in Section 6.

2 Related Work

DL popularity is steadily increasing thanks to its impact on many application domains (ranging from image and voice recognition to text processing) and has received a lot of interest from many academic and industrial groups. Advances are boosted by enhancements of the deep network structure and learning process (e.g., dropout Srivastava et al. (2014), network in network Lin et al. (2013), scale jittering Vincent et al. (2010)) and by the availability of GPUs, which provide up to 40$\times$ reduction over CPU systems Bahrampour et al. (2015).

Over the last few years, several frameworks have been developed and are constantly extended to ease the development of DL models and to optimize different aspects of training and deployment of DL applications. The work in Bahrampour et al. (2015) provides a comparative study of Caffe, Neon, Theano, and Torch, by analyzing their extensibility and performance and considering both CPUs and GPUs. The paper provides insights into how performance varies across different batch sizes and different convolution algorithm implementations, but it does not provide means to generalize performance estimates to different settings.

Since common NNs are deployed on GPUs, it is important to understand how such hardware can influence performance, but unfortunately only a few studies are available in the literature.

Jia et al. (2012) propose the Stargazer framework to build performance models for a simulator running on GPU, so as to correlate several GPU parameters to the simulator execution time. Given the daunting size of the design space (which considers very low level parameters like the number of thread blocks concurrently running on one core or how many intra- and inter-warp memory requests may be coalesced into one memory access), they exploit sparse random sampling and iterative model selection, thus creating step by step an accurate linear regression model. Another approach to the issue is proposed in Liu et al. (2007), where the authors elaborate a detailed analytical model of general purpose applications on GPUs. The proposed model consists of three general expressions to estimate the time taken for common operations, according to their dependence on data size or computational capabilities.

Similar *analytical* modeling approaches Baghsorkhi et al. (2010); Zhang and Owens (2011); Hong and Kim (2009); Song et al. (2013) rely on micro-architecture information to predict GPU performance. As GPU architectures continue to evolve, the main issue of analytical models is that a minor change in the architecture may require extensive work to adapt the model to hardware enhancements Dao et al. (2015).

Given the complexity of GPU hardware (due to the large number of processors, the thread context switching mechanism, and the on-board hierarchical memory subsystem), recently black box approaches based on ML are favored over analytical models. Indeed, black box approaches can derive performance models from data to make predictions without a priori knowledge about the internals of the target system. On the other hand, ML models Dao et al. (2015); Barnes et al. (2008); Bitirgen et al. (2008); Kerr et al. (2010); Lu et al. (2017); Gupta et al. (2018); Peng et al. (2018); Dube et al. (2019) require to perform an initial profiling campaign to gather training data to learn the mapping among the features of an application and its execution time. An overview and quantitative comparison among recent analytical and ML-based model proposals is reported in Madougou et al. (2016).

The contribution closest to this work can be found in Dao et al. (2015). The work considers the OpenCL benchmark suite and obtains ϵ-SVR models able to predict each

OpenCL kernel performance with an average 5–10 % error. However, the approach is rather low-level since it requires to instrument each kernel's code and to identify through an ad-hoc profiling activity low-level features such as the maximum number of active work groups, the number of registers used per work item, the number of branches and divergent branches per work item. In this paper, the achieved accuracy is similar to Dao et al. and is obtained by processing only DL application framework logs. Moreover, the maximum error obtained is also significantly lower, about 27 % in the worst case, while more than 70 % error is obtained by Dao et al. On the other hand, the proposed approach focuses on estimating NNs training time, while the usage of low-level features allows the technique proposed by Dao et al. to predict the performance of other classes of applications.

Kerr et al. (2010) profile and build models for a range of applications, run either on CPUs or GPUs, exploiting the Ocelot framework Diamos et al. (2010). Relying on 37 performance metrics, they exploit principal component analysis and linear regression in order to highlight those features that are more likely to affect performance on heterogeneous processors. The approach is rather general since the Ocelot infrastructure allows for simulating GPUs, gathering performance metrics, and instrumenting kernels. The work models CUDA kernels' performance using only metrics that are available before a kernel is executed. Unfortunately, the Ocelot infrastructure is not actively developed anymore, limiting its support to old GPU generations. Along the same lines, Luk et al. (2009) describe Qilin, a technique for adaptively mapping computation onto CPUs or GPUs, depending on the application as well as system characteristics. With this approach, the authors show an improved speedup with respect to manually associating jobs and resources. Finally, Gupta et al. (2018) propose an online modeling approach to predict the single frame processing time starting from GPU frequency and performance counters for mobile devices.

In the DL area, Hadjis et al. (2016) present solutions to minimize the total training time of CNNs, given the network architecture of the DL model, the data set used for training, the set of available computational resources and their throughput, and the network speed. The framework is able to identify the optimal split of the DL model across multiple compute groups and the optimal number of servers per group. They also propose a solution to improve the stochastic gradient descent (SGD) momentum update to compensate the staleness introduced by the adoption of multiple compute groups. However, their system cannot predict a priori the execution time of a given number of iterations nor the training time in experiments with different batch size.

A higher level approach, which considers matrix multiplication computation time as the core feature, is presented in Lu et al. (2017), where the authors focus on the deployment of CNNs on mobile devices. According to this consideration, they focus only on the forward pass, whose overall execution time is predicted starting from an estimate of the time taken by matrix multiplications when the CNNs are deployed on a CPU or a GPU. This approach entails extracting tensor sizes from network specifications and associating them to their expected response times, according to platform-specific benchmarks. In the end, the approach is more accurate when CPUs are considered (the percentage error they obtained is in the range 6–15 %, while they obtain 16–21 % for the GPU case). Similarly, in our previous work Gianniti et al. (2018b), we derive linear regression models for estimating CNN training times by relying on the layers computational complexity in terms of simple primitives available on GPUs. This enables performance prediction even on networks never taken into account during the learning phase. However, our initial study is limited to CNNs trained on a single GPU with the Caffe framework. In a more recent work Gianniti et al. (2019), we compared our per layer model in Gianniti et al. (2018b) with a pure black box ML end-to-end model but the study was still limited to a single DL framework and could not generalize prediction to different GPU hardware. Finally, the work in Dube et al. (2019) proposed AI Gauge, a framework based on ML where models are continuously calibrated processing job traces. The proposed models achieve less than 10% relative error on average, but, however, are limited to single GPU deployments.

The main novelty of this paper with respect to previous literature contributions lays in the generality of the proposed models and in their extrapolation capability. Each NN training is associated with a single model that allows users to predict the execution time on new target architectures. Different from previous approaches, the proposed performance models do not require profiling nor simulation of the application on the new target architecture: GPU data sheets provide all the relevant information. Moreover, several scenarios of interest can be investigated with good accuracy and limited profiling, such as evaluating how the training time changes by changing the number of iterations and batch size or by varying the number of inner modules in the networks.

3 Models Training

To estimate the execution time required for training an NN, a linear model that correlates some features of the training process with its execution time is considered. The model is built using multiple linear regression: Madougou et al. (2016); Lee et al. (2007) show how linear regression-based approaches usually perform better than other solutions (for instance, neural networks) in terms of accuracy for modeling the performance of massively parallel applications.

This section is organized as follows. Section 3.1 overviews the set of NNs considered in this study. Section 3.2 describes the initial set of features, which are considered, combined, augmented, and finally selected according to the approach described in Section 3.3. Lastly, Section 3.4 introduces the analyses devised to evaluate the performance prediction models' generalization capabilities.

3.1 Types of Neural Networks

In the following, three CNNs and one RNN will be considered, namely, AlexNet Krizhevsky et al. (2012), VGG Simonyan and Zisserman (2014), ResNet He et al. (2016), and the Baidu DeepSpeech network implemented by Mozilla Foundation (2019b). These three CNNs for image classification tasks have been selected since they have been widely studied, brought important advancements in general network architecture, and are frequently used in transfer learning applications, when the time to train a network and/or the labeled training images are limited Csurka (2017). Moreover, since their architectures are heterogeneous, they can be considered representative samples of the variety of architectures used in practice for image classification and video processing.

The Mozilla DeepSpeech implementation for audio file transcription has been selected as a relevant representative of the RNN architecture family, since it implements several optimizations with respect to the initial Deep Speech version 1 Hannun et al. (2014) and version 2 Amodei et al. (2015) papers, including the introduction of long short term memory (LSTM) cells in the recurrent layer, it is able to achieve close to human accuracy and it can be used in production to determine transcripts from streaming audio with off-the-shelf hardware Morais (2019).

AlexNet is particularly important from the historical perspective, since in 2012 it won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) achieving a top five test error rate of 15.4 %, while the next best entry achieved an error of 26.2 %. This result was a very significant improvement and since then CNNs became commonplace in the competition. AlexNet introduced rectified linear units as nonlinear functions rather than the hyperbolic tangent, thus achieving a significant reduction in training time, and has a relatively simple layout compared to modern architectures, as it includes only five convolutional, five pooling, five dropout, and three fully connected layers. Nowadays, AlexNet can be considered representative of small CNNs. A small complexity has not only a significant impact on the throughput of the CNN and so on its overall training time. The training process, indeed, is composed of two activities: data loading and weights update. At each iteration, images are loaded into GPU memory and then weights are updated according to the training algorithm, e.g., SGD or Adam. In complex deep CNNs, the data load delay is negligible with respect to the actual training process. On the contrary, in the case of AlexNet, this contribution is significant and it has to be taken into account.

VGG was proposed at ILSVRC 2014 and provided a 7.3 % error rate, even if it did not win the accuracy competition. The main characteristic of VGG is to adopt a fixed module based on an architecture with very small (3×3) convolution filters, which is then replicated up to 19 times. VGG demonstrated that a significant improvement on the prior-art configurations can be achieved by pushing the network depth. As a drawback, the number of parameters grows up to more than 140 million, significantly increasing the required memory. A year after, a new record in terms of the error rate (only 3.6 %, similar to humans performance He et al. (2016)) was achieved by ResNet. The most important innovation introduced by ResNet is the adoption of residual networks, which demonstrated easier to be optimized and can gain accuracy from considerably increased depth (the best performance was achieved by stacking up to 152 layers). Despite the incremented accuracy, the number of parameters in ResNet is smaller than VGG's (about 60 million for the version with 152 layers). This significantly reduces its memory usage and makes it suitable for training on smaller GPUs, as Section 4 shows.

A common characteristic between VGG and ResNet is to exploit a repeated basic structure, whose number of replicas can be increased to improve the DL model accuracy, thus obtaining production quality models. The layers across VGG and ResNet follow two main design rules He et al. (2016): (i) for the same output feature map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer. In this way, modern nets are characterized by layers requiring about the same computational effort.

Finally, the Mozilla open source implementation of the Deep Speech network has been chosen since, even if it is optimized for multi-GPU training, the adopted communication and synchronization scheme makes the performance modeling more challenging.

Deep Speech includes five layers: the input audio file is fed into three fully connected layers, followed by a bidirectional RNN layer based on LSTM, and finally a fully connected layer. The main limitation introduced by the recurrent layer is that its hidden units are computed sequentially: the RNN forward propagation requires to be computed from the first time-slice of the input audio and each forward computation step depends on the previous one. Instead, backward propagation starts computing from the last time-slice, with an inverse dependency. The solution proposed in Hannun et al. (2014) to train the Deep Speech network is to divide the recurrent layer by the time dimension, after that it is possible to switch the roles of the GPUs at half time $T/2$ of the audio speech and to exchange the intermediate activations. In this way, half of the GPUs that started the computation of the forward RNN propagation will end up computing the backward propagation, and similarly for the other GPUs, avoiding massive data transfers. Furthermore, the network model weights are stored in the CPU memory and a synchronous gradient optimization scheme is adopted. Once a mini-batch is sent to each GPU, GPUs compute their contribution to the gradient, which is sent back to the CPU. The CPU waits until all the working GPUs end the computations on top of their mini-batches and, finally, computes the mean gradient and updates the model, thus introducing a strict synchronization barrier. Since audio clips are characterized by different length, this latter synchronization step leads to performance degradation due to an unbalanced load of individual GPUs (at every step the CPU has to wait for the slowest GPU). Moreover, the language model is also introduced in the deep network by introducing the CTC (Connectionist Temporal Classification) decoder initially proposed in Maas et al. (2014).

3.2 Initial Feature Set

The training time of an NN is mainly characterized by two sets of features: a set describing the characteristics of the training process to be executed and a set describing the characteristics of the hardware that will perform such a process. The features belonging to the former group are:

– I: The number of iterations executed during the training.
– B: The number of training examples processed during an iteration, i.e., the batch size; in case of more than one GPU, the batch size identifies the overall number of training examples processed by all the GPUs (following the strong scale approach PyTorch (2018)).

The main aim of this work is to provide a methodology to predict the performance of the analyzed application when executed on new hardware. For this reason, a different performance model is built for each pair of framework-trained NN, and, in the scenarios where the topology of the network is fixed, it is not necessary to add any features describing the structure of the network, since this will be the same in all the experiments used to build the model and estimated by means of it. Moreover, since the implementations of CNNs rely on fixed size images, obtained by cropping original images, the image size is not added as a model feature. On the contrary, in the scenarios where the topology of the network is not fixed, additional features are added to describe it. In particular, for what concerns the scenarios with ResNet with variable depth, an additional feature is added to describe the structure of the trained CNN: N, i.e., the number of inner modules of the deep net.

The above-listed features describe the characteristics of a NN training process independently from the hardware on which it will be run. Training a different performance model for each target GPU could potentially increase the accuracy of the estimate, but would prevent prediction on new hardware. On the contrary, a single model that includes features describing the hardware is used to predict NN training time on different GPUs. The features used to describe hardware characteristics are:

- P: The computational power of the used GPUs, measured in single precision GFlops/second.
- G: The number of GPUs used.
- T: The number of CPU threads used to load in parallel the training examples of a mini-batch to GPU memory.
- D: The disk delay, measured as the time required to load 120 k files of 192 kB from disk into the memory of a GPU. With respect to the nominal speed of the disk, this value has two main advantages: (i) it better takes into account the disk access pattern of the initial data load step, which is performed before the deep net training starts; (ii) it can be collected by means of a simple benchmark, without requiring detailed knowledge of the hardware. Moreover, note that obtaining detailed storage and I/O hardware characteristics in case of cloud deployments may, indeed, not be possible.

The described D requires direct access to the target platform. Since this is not always possible, approximations can be obtained by considering VMs of the same family or by considering nominal characteristics of the target hardware. For example, if the information about the D measured on a Microsoft Azure NC6 VM is available and the D of a Microsoft Azure NC12 (which differs from the former for the number of CPUs and GPUs) is not, the latter value can be approximated to the former.

3.3 Feature Selection

Some of the previously mentioned features can be expected to directly contribute to the observed deep net training execution time. For instance, increasing the number of performed iterations intuitively suggests an increase also in the NN training time. On the other hand, there are some metrics that more appropriately should appear in their reciprocals: for example, G and T (which represents the number of parallel executors in charge of a training process) are likely to produce terms that depend on their inverse Gianniti et al. (2018a). In this paper, the feature set is augmented with all the reciprocals of the original features, so as to cater both for those parameters that intuitively contribute in this way and for possible second-order effects, which may be harder to anticipate. In principle, it would be possible to use arbitrary nonlinear transformations of the original features in creating the complete set, such as higher degree powers or elementary functions. In this research work only measures coming from the system and their reciprocals are considered. This choice is aimed at bounding the size of the initial available feature set. The attained accuracy justifies this choice a posteriori.

Alongside adding reciprocals, another relevant approach to augmenting the feature set is the use of interaction terms. As a motivating example, the product $I \cdot B$ yields the total number of processed training examples, hence a relevant contribution for an accurate performance prediction model. In line with this method, the feature set is also extended with crossover terms. Each term can contain any combination of original features and reciprocals under the constraint that every monomial has at most degree 1 in every variable. This constraint comes from the previously mentioned choice of not using any higher degree powers. Additionally, combined terms cannot contain both a feature and its reciprocal, so as to avoid adding redundant information and/or introducing numerical issues.

In order to obtain reasonably sized models, this paper applies a simple feature selection technique based on the statistical properties of linear regression. Draper and Smith (1966) outline an approach that tries to merge the basic forward selection and backward elimination schemes, thus achieving the best of both worlds. On one side, backward elimination starts with the full set of features and iteratively drops the least significant term from the regression equation, eventually stopping when all the remaining coefficients are significant at a given confidence level. In a dual fashion, forward selection starts with an empty model and iteratively adds the most promising features one at a time, until the latest addition results to be not significant enough. In both cases, the decision is made based on the p-value of single coefficient t-tests, where at each iteration the least significant (respectively, last added) feature is compared to a predefined threshold. Since both methods are somewhat impaired by their greedy approach, Draper and Smith suggest to start with an empty model and proceed in combined steps that try both to add a feature and to remove one, until neither succeeds at the preliminarily set up confidence levels Draper and Smith (1966).

The decision of adopting this stepwise selection technique is driven by its simplicity, despite the lack of a validation set. Nonetheless, the experimental results suggest that this fact does not harm accuracy. In particular, Section 5.4 shows how the final models retain good generalization capabilities.

3.4 Extrapolation Analyses Setting

A most relevant aspect of this research work is investigating the performance prediction models' generalization capabilities. It is of the utmost importance quantifying to what extent the equations obtained via linear regression remain accurate if applied outside of the feature range spanned by training data. This aspect has a significant practical implication: generalizable models enable accurate performance prediction in conditions that users may not be able, or willing, to empirically investigate. This section elaborates on extrapolation scenarios.

The most obvious scenario is the extrapolation on the number of iterations, I, which simply corresponds to training a NN for more epochs. Such information helps in scheduling jobs on a shared infrastructure or can be used, conversely, to tune the number of epochs so as to fit a given time window. Similarly, extrapolating on the number of inner modules, N, allows for fine-tuning the network depth of the architectures that can be parametrized in this sense, such as ResNet and VGG.

Another set of features can have an even higher economic impact, as their interpretation is linked with hardware changes in the deployment of interest. For instance, since it is common practice to use the largest batch size that fits on the GPUs at hand, in order to reach the optimal parallelism, extrapolating on B corresponds to using different GPUs with larger memory. Along the same lines, extrapolation on computational power P means switching to GPUs with a higher nominal speed in GFlops/second. In the end, extrapolating on the number of data loading threads T or of GPUs G relates to the installation of more, respectively, CPUs or graphics cards.

Table 1: Characteristics of the target machines

Machine Type	(v)CPUs	Mem. [GB]	GPU type	Computational Power [GFlops/s]	N. GPUs
Azure Standard_NC6	6	56	K80	5591	1
Azure Standard_NC12	12	112	K80	5591	2
Azure Standard_NC24	24	224	K80	5591	4
Azure Standard_NV6	6	56	M60	7365	1
Azure Standard_NV12	12	112	M60	7365	2
Azure Standard_NV24	24	224	M60	7365	4
In-house server 1	20	48	Quadro P600	1195	2
In-house server 2	40	256	GTX 1080Ti	11339	8

4 Experimental Setup

This section describes in detail the experimental setup for collecting data and the set of conducted deep net profiling experiments. To enforce the generality of the proposed approach, different open source frameworks and multiple NNs have been considered. The adopted frameworks are *PyTorch 0.3.1* Paszke et al. (2017); PyTorch (2018) and *TensorFlow 1.8.0* Abadi et al. (2016); TensorFlow (2018), while the trained CNNs are AlexNet Krizhevsky et al. (2012), ResNet-50 He et al. (2015), and VGG-19 Simonyan and Zisserman (2014), whose implementations are already available within the considered frameworks. As a representative of RNNs, the DeepSpeech Hannun et al. (2014) implementation provided by Mozilla has been selected. The implementation is available only for TensorFlow, while a corresponding PyTorch implementation is not available.

The experiments have been run on four different types of machines, whose characteristics are summarized in Table 1:

- *Microsoft Azure NC*: They are the cheapest Microsoft Azure VMs based on GPUs, including up to four K80 GPUs.
- *Microsoft Azure NV*: Microsoft Azure machines based on up to four M60 GPUs.
- *In-house server 1*: An in-house server that includes an Intel Xeon Silver 4114 and two Quadro P600 GPUs.
- *In-house server 2*: An in-house server based on two Intel Xeon E5-2630 v4 and 8 Geforce GTX 1080Ti GPUs.

VGG-19 cannot be trained on the *in-house server 1* since its parameters do not fit in GPU memory. To test the generalization capabilities of the performance models when the number of inner modules is varied, ResNet[1] has been considered as reference deep network and the number of inner modules has been varied between 1 and 10.

The addressed scenario for CNNs is the classification of images belonging to the *ImageNet* Deng et al. (2009) database. The images are cropped to 32x32 pixel when used as input of the ResNet with variable number of inner modules and to 224x224 in all the other scenarios. To speed up the experimental campaign, the set of analyzed images is a subset composed of around 120,000 items evenly partitioned into 100 classes. DeepSpeech has been trained by considering 512 samples from the Common Voice open source corpus Foundation (2019a) including English language voice records.

As in other research studies, each experiment has been run immediately after preliminary one-epoch-long experiments and (see, e.g., Hadjis et al. (2016)) the execution

[1] An implementation of ResNet with a variable number of inner modules is available at `https://github.com/KellerJordan/ResNet-PyTorch-CIFAR10`.

time of the first 20 iterations has been excluded from each experiment's total time, because they can take significantly longer to complete than the subsequent iterations. It is worth noting that their removal does not affect the significance of the trained performance models, since in real scenarios the first 20 iterations are negligible with respect to the full training, which runs for several thousands of iterations. To remove other possible "warm-up" effects, each experiment performed on cloud VMs has been repeated at least three times and the data about its first run discarded. In this type of scenario, indeed, the whole first epoch can have a significant time delay caused by the retrieval of the training examples used during training, which may not be immediately available on the VM's local disk. Even in this case, the impossibility of correctly estimating the first epoch of a training procedure is not a significant limitation, since this is only a limited fraction of the overall deep net training process. Moreover, to reduce the overall execution time of the experimental campaign, the upper bound of the number of epochs has been set to three. To verify that the data collected on the first three epochs can be effectively used to build general models able to predict the execution time of real training processes with thousands of epochs, some ad-hoc long-running experiments have been performed. In particular, for each framework, for each network, and for each type of GPU but GTX 1080Ti, a long experiment (at least 24 hours) has been run. Long runs on GTX 1080Ti could not be performed because of the limited availability of in-house server 2.

To increase the number of available samples, for every run on the target system, timing data is also collected after 25 %, 50 %, and 75 % iterations, respectively. In this way, four data points are extracted from every single run. Finally, to mitigate the effects of system perturbation (for example, running of operating system services) on the collected data and to improve the accuracy of the generated models, all the experiments whose execution time is shorter than 10 seconds have been removed from the data sets.

Different ranges of batch size have been considered for different networks and for different machines, since GPU devices with larger memory support training with larger batch sizes. A strong scale approach has been adopted: the batch size identifies the overall number of training examples processed, possibly across multiple GPUs, during an iteration PyTorch (2018). The largest batch size is 8,192 for AlexNet, 512 for ResNet-50 and VGG-19, and 4,096 for DeepSpeech, when running on 8 GTX 1080Ti. Several values of numbers of CPU threads have been analyzed for the AlexNet experiments, while only up to 8 CPU threads have been used in the training of ResNet-50, VGG-19, and DeepSpeech since varying the thread numbers did not significantly influence the forward or backward times of these NNs. The overall number of deep net trainings is about 10,400 with an overall execution time of more than 5,500 hours. For the sake of completeness, the size of the training set for each investigated scenario is reported in the appendix. Note that the reported numbers refer to available samples. The number of corresponding runs is roughly one fourth, i.e., for each experiment four samples are generated as previously described. Some of the generated samples, however, have been discarded because shorter than 10 seconds.

Finally, the accuracy of the models is evaluated considering their mean absolute percentage error (MAPE) on the test set:

$$MAPE = \frac{100\%}{S} \sum_{k=1}^{S} \left| \frac{y_k - \hat{y}_k}{y_k} \right| \tag{1}$$

Table 2: Relative percentage difference between average iteration time of initial epochs and average iteration time over the whole long running experiment

Network	Framework	N. Initial Epochs	GPU Type P600	K80	M60
AlexNet	PyTorch	1	0.73	7.42	9.25
		3	0.18	1.52	1.38
	TensorFlow	1	0.18	7.14	4.76
		3	0.12	7.40	4.80
ResNet-50	PyTorch	1	0.08	1.02	0.24
		3	0.49	0.20	0.01
	TensorFlow	1	0.33	0.27	0.62
		3	0.31	0.09	0.82
VGG-19	PyTorch	1	-	2.71	0.78
		3	-	1.50	0.22
	TensorFlow	1	-	4.00	3.62
		3	-	3.91	3.72
DeepSpeech	TensorFlow	1	0.48	3.43	6.09
		3	0.79	2.81	3.36

where

- S is the number of samples in the test set.
- y_k is the training time measured on the operational system.
- $\hat{y}_k$ is the predicted training time from the learnt model.

5 Experimental Results

In this section, the results obtained in training the performance models for the different networks and for the different frameworks will be presented. It is worth noting that, in order to have models suitable to be exploited in what-if performance analyses and capacity planning, their prediction error should be small. As from the common practice Lazowska et al. (1984), we will consider a model accurate if its MAPE is lower than 30 %.

Before presenting accuracy results obtained with the built models, Section 5.1 analyzes if data collected on the initial epochs of a training can be effectively used to predict long-running experiments. Next, Section 5.2 presents the results of the trained models in predicting the performance of a specific network when a single framework is considered, but multiple types and numbers of GPUs are available. Section 5.3 analyzes more in depth the trained performance models properties and shows the accuracy loss when one of the features identified in Section 3.2 is removed from the input data. Section 5.4 presents the results obtained when the trained models are used to perform extrapolation on batch size, number and type of GPUs, and depth of the network. Finally, Section 5.5 analyses the results we achieved and discuss the limitations of our approach.

5.1 Long Runs Analysis

In this section the data collected on the long runs described in Section 4 is presented. The aim of this ad-hoc set of experiments is to verify that the execution time of

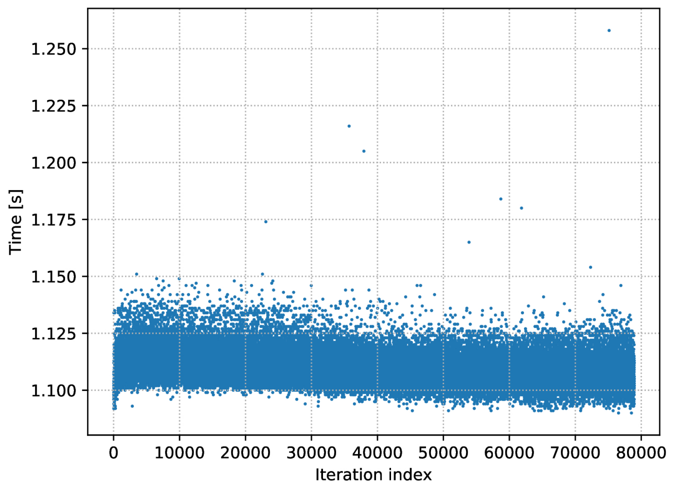

Figure 1: Execution time of iterations of ResNet-50 on TensorFlow with 4 K80.

individual epochs is stable so that the models trained by using data on a few epochs can be effectively used to predict performance of long-running deep net trainings.

As an example, Figure 1 plots the execution time of each ResNet-50 iteration on TensorFlow with four K80 over 24 hours. The execution time of most of the iterations is in the range 1.09 to 1.13 seconds, while the maximum execution time is 1.27 seconds. Table 2 presents the difference between the average execution time of an iteration in the whole long-running experiment and the average execution time of one iteration computed either on the first epoch or on the first three epochs. Because of limited time slots accessibility, results on the in-house server 2 could not be collected for multiple day executions. On the in-house server 1 the difference between the time of the initial epochs and the average execution time of epochs for long experiments is very small (less than 1 %). Results obtained on the Azure VMs are characterized by a larger difference, but in all the scenario the difference is below 10 %. Moreover, when the number of initial epochs is 3, the difference is reduced to less than 8 %, so short experiments data can be effectively used to train models to predict real scenario deep net training execution times. It is worth noting that possible “warm-up” issues have been removed by the preliminary one epoch long experiments.

By analyzing individual deep nets, it can be noticed that the difference for ResNet-50 and VGG-19 is not larger than 4 %, while, on the contrary, AlexNet differences are more significant, up to 9.25 %. This is caused by the impact of data loading on AlexNet, which has been described in Section 3.1: since this is more subject to system perturbations, its variance is more relevant, resulting in larger differences among the epochs execution time.

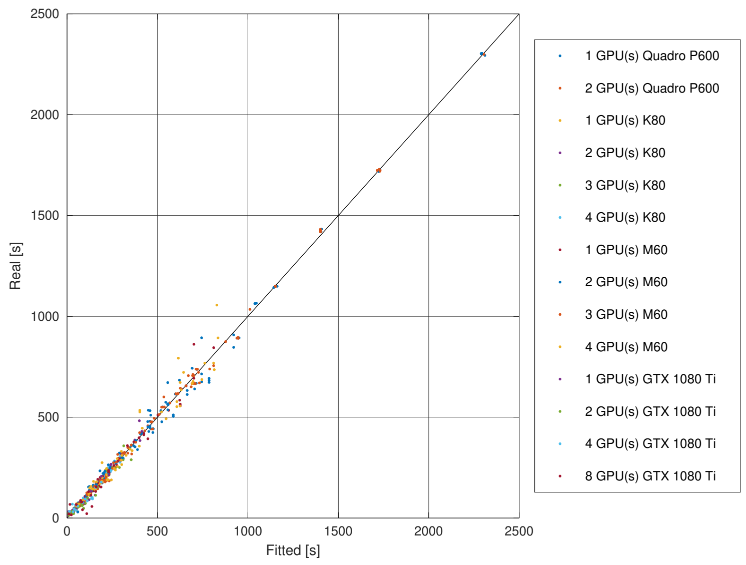

Figure 2: Real vs. predicted values for AlexNet/TensorFlow

5.2 Hold-out Results

Hold-out models are trained by exploiting the data of all the performed experiments with a given framework and a given network on different numbers and types of GPUs. The set of available experiments have been randomly split into a train set (containing 80 % of the samples) and a test set (including the remaining 20 % of the samples). The set of initial features includes all the ones described in Section 3.2. Features are then augmented and selected by the stepwise algorithm presented in Section 3.3. Performance models accuracy is evaluated using MAPE.

As an example, Figure 2 shows the execution times measured on the target systems for the TensorFlow implementation of AlexNet on the test set versus the predicted execution time. The diagonal represents exact estimations: the farther a point is from this line, the worse is the performance model prediction. Moreover, the points that lie below the line represent overestimated execution times, while, on the contrary, points that are above are representative of optimistic predictions. The obtained MAPE is less than 11 %: as can be noticed, most of the AlexNet training times are quite accurately estimated and there is not any bias towards their over or underestimation. Figure 3 shows instead the results obtained in a different scenario: ResNet-50 trained by means of PyTorch. In this case, the error is quite small (less than 3 %) since most of the experiments are very accurately estimated. Overall, these results are very good and are promising for investigating extrapolation scenarios.

For the sake of brevity, a summary of the results obtained on the different NNs is reported in Table 3. The rightmost columns present the MAPE obtained on the test set with different frameworks and networks. The results obtained with the two frameworks are quite similar: estimating the training time for the AlexNet network is more difficult than the others because of the data load effect described in Section 3.1.

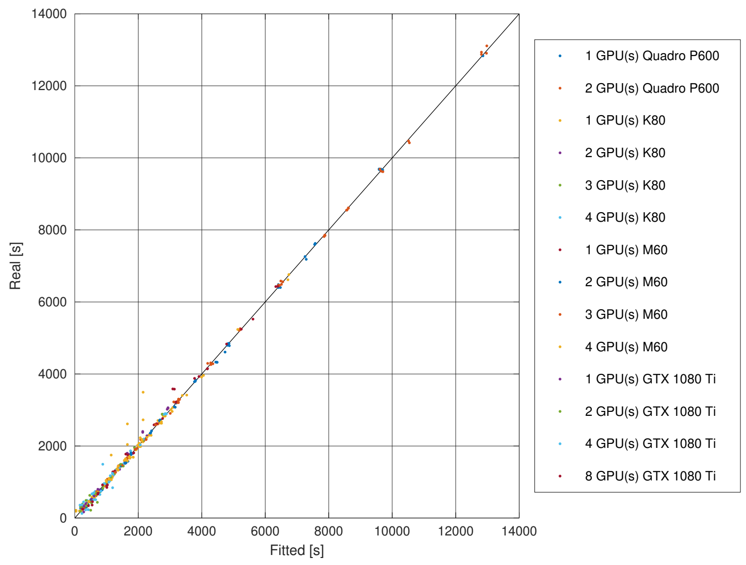

Figure 3: Real vs. predicted values for ResNet-50/PyTorch

Table 3: MAPE [%] on test set for different models

		Number of used features			
Network	Framework	All	15	10	5
AlexNet	PyTorch	8.70	11.62	12.04	19.41
	TensorFlow	10.95	13.07	14.61	19.22
ResNet-50	PyTorch	6.50	16.86	15.95	16.98
	TensorFlow	6.43	13.75	13.44	22.05
VGG-19	PyTorch	5.24	11.72	12.04	15.10
	TensorFlow	4.18	5.69	5.96	7.39
DeepSpeech	TensorFlow	8.51	15.44	17.70	22.57

On the contrary, the regularity of the ResNet-50, VGG-19, and DeepSpeech networks results in very accurate performance estimation models.

The *All* column of Table 3 reports the accuracy of the models built exploiting all the features identified as significant by the stepwise feature selection algorithm described in Section 3.3. The remaining columns present the accuracy obtained with a limited number of features, namely, the top 15, 10, or 5 variables identified via the selection algorithm.

Not all the added features provide the same amount of information for describing the performance of NN training. Since the considered stepwise approach selects features in an iterative way, it is possible to evaluate how the accuracy changes when adding the features incrementally. Figure 4 shows how the prediction error of a model decreases when the number of used features increases. It is worth noting that there are very significant improvements in the accuracy of the model up to the sixth added feature. After that, adding more features improves the model accuracy only slightly. Nevertheless, considering all the selected features instead of a limited number still results in a significant accuracy improvement.

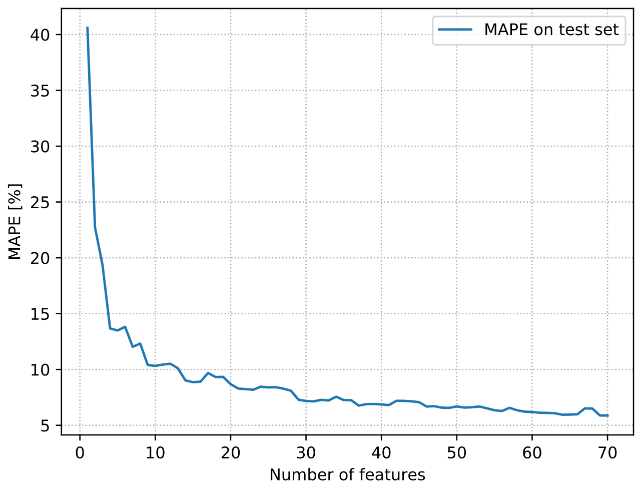

Figure 4: AlexNet/PyTorch MAPE against varying number of features

The rightmost columns of Table 3 report the accuracy of the performance models trained by limiting the number of used features to 15, 10, and 5. In particular, comparing the results achieved by the models exploiting all the features with respect to the models built with only 5 features, one can observe the level of increased accuracy provided by the other features. The smallest absolute gain is obtained on VGG-19 with TensorFlow, where the MAPE decreases from 7.39 % to 4.18 %, while, on the opposite, the largest absolute gain is obtained on ResNet-50 with TensorFlow, where the MAPE decreases from 22.05 % to 6.43 %.

5.3 Features Relevance Analysis

In Section 3.2 the set of features useful to train NN performance models for the considered nets and frameworks have been identified and initial analysis of the trade-off between models accuracy and size has been performed. This section analyzes the same set of scenarios considered in the previous section, but performance models are trained by excluding one feature (and all its derived terms) at a time, so as to assess the impact of individual features on performance prediction accuracy.

For each network and for each framework, five different performance models are trained. The first model is trained by removing the number of iterations (I) and all the derived terms. The obtained error is very large (more than 75 % in all the cases) as expected since the number of iterations obviously has a major impact on the deep net training time. Moreover, the number of iterations is definitely the feature with the largest possible range. In the considered experiments, to reduce the overall experiment time, this value has been limited to a few thousand. In real scenarios, its value can be much larger (for instance, up to 600 k in the experiments described in He et al. (2015)), so ignoring it in performance model training would result in a much larger error.

Table 4: MAPE [%] on test set of models built excluding features

		Features excluded during model training					
Network	Framework	None	I	B	G	T	P, D
AlexNet	PyTorch	8.70	100.57	50.88	12.81	59.07	20.39
	TensorFlow	10.95	135.80	73.47	33.88	21.23	32.81
ResNet-50	PyTorch	6.50	99.97	38.56	30.60	10.92	28.52
	TensorFlow	6.43	78.29	44.71	30.56	8.57	41.52
VGG-19	PyTorch	5.24	84.97	33.46	20.56	6.15	52.62
	TensorFlow	4.18	85.00	30.59	14.74	4.15	24.64
DeepSpeech	TensorFlow	9.85	170.45	108.47	24.50	10.72	10.53

The range of the possible values of batch size (B) is instead mainly bound by the maximum memory available on the GPUs and it differs according to the characteristics described in Section 4. The results reported in the column B of Table 4 show how even this feature (and its derived terms) are meaningful in describing the performance of NN training. The performance models for DeepSpeech have the largest errors mainly because these experiments are characterized by a wider range for this feature with respect to other networks. The errors on AlexNet are larger than ResNet-50 and VGG-19 since the batch size influences data loading time, which has significant relevance in the training of this type of networks. The errors of the other models trained without information about batch size is smaller, but still quite significant. Even in the best case, removing this type of information decreases the accuracy of the model by more than 5 times: in the case of VGG-19 implemented with PyTorch the prediction error grows from 5.24 % to 33.46 %.

The third set of performance models are trained by removing the number of GPUs (G) and present smaller prediction errors with respect to the first two classes of models. The models for AlexNet are characterized by an even smaller decrement of accuracy. This reduced increase in the error is motivated by two main reasons: the impact of data loading and the set of experiments that has been considered. For the former reason, because of the small computational complexity, the data loading time provides a significant contribution to the overall training time of the network. The increment of the number of GPUs does not provide any benefit on the loading time: on the contrary, contention in accessing storage resources can decrease device efficiency. Concerning the set of performed experiments, only a limited set of them fully utilize system resources (i.e., use the maximum value of CPU threads, batch size, etc.). For example, in the experiments exploiting two CPU threads to load data, most of the time is spent in this process: incrementing the number of GPUs only speeds up a limited portion of the entire training process, resulting in a very small overall performance gain.

For the same reason, the number of CPU threads (T and its reciprocal) used to load training examples into GPU memory is a significant feature only for AlexNet. Since the data loading time is a significant fraction of the overall time, incrementing the number of CPU threads devoted to this aim can significantly improve the performance of the training process. Training a performance model that does not take this aspect into account results in a significant estimation error (62.19 % for PyTorch and 21.23 % for TensorFlow). On the contrary, in complex networks like ResNet-50 and DeepSpeech, the data loading time is less relevant, so the effect of using more CPU threads is less significant and, as a consequence, the performance models trained without this feature do not present a relevant accuracy decrease. Finally, for VGG-19, which is characterized by an even higher computational complexity, there is not any significant contribution provided by the CPU threads data.

The last column of Table 4 reports the results of performance models trained without considering the features describing hardware characteristics (i.e., P—GPU computational power, and D—disk performance). In the experimental campaign of this paper, these two features are highly correlated. Given a specific GPU type, in most of the (virtual) machines, this is always associated with disks boasting the same characteristics. Because of the correlation, in most of the cases removing just one of the two features does not produce a significant degradation of the trained model accuracy. The accuracy of the performance model for DeepSpeech does not change when hardware related features are removed. On all the considered GPUs but Quadro P600 computational power is not the bottleneck of the system. This can be expected since the memory access and the CPU-GPU communication are usually the training process bottlenecks Amodei et al. (2015), due especially to the beam search that involves repeated lookups in the n-gram language model, most of which translate to uncached reads from memory and to the update of the model performed at the end of each iteration. On the contrary, the relevant feature is the amount of available memory, but this determines what are the experiments that can be run on a specific system (i.e., which is the maximum batch size) and not what is the execution time to perform the training. The second smallest increment of errors occurs for AlexNet, mainly due to the removal of D. It is worth noting that removing hardware-related features does not introduce a very large error (20.71 % for Pytorch and 32.81 % for TensorFlow). The reasons for this small increment are the same as previously described in the case of removal of G. Moreover, the relative range of this feature is even smaller. In the target machines considered for the experimental campaign, the gap between the slowest and the fastest disk is less than two.

The model without disk information roughly assumes a disk with the average performance among the considered ones, with a limited approximation with respect to the actual data.

The errors of the models without hardware information for ResNet-50 and VGG-19 are larger and are mainly caused by the removal of the computational power feature associated with the GPUs. For this type of feature, the range is larger: the most powerful GPU considered in the experimental campaign is around 10 times faster than the least powerful. This gap reflects on the error of the performance models trained without P: In the worst case (VGG-19 on PyTorch) the error is 52.62 %.

5.4 Extrapolation Results

The main aim of a performance model is to provide a means to predict an application execution time (deep network training time in this paper settings) in setups for which profiling data is not available. Two main scenarios can be of interest:

- The exploitation of new hardware, both in terms of cloud VMs or new in-house servers.
- The training of new versions of a NN.

This section investigates the ability of the training performance models to accurately estimate these new scenarios. In particular, the results will focus on the extrapolation capability presented in Section 3.4. Extrapolation is performed in one dimension:

Table 5: MAPE [%] on test set of batch size extrapolation models

		GPU Type and Number					
		P600		K80			
Network	Framework	1	2	1	2	3	4
AlexNet	PyTorch	11.12	5.33	1.74	3.33	1.81	0.66
	TensorFlow	9.83	10.04	2.30	2.61	4.28	2.82
ResNet-50	PyTorch	10.64	11.97	0.76	7.83	3.09	4.53
	TensorFlow	10.64	11.97	10.25	1.27	1.84	6.83
VGG-19	PyTorch	-	-	13.88	21.71	27.63	9.65
	TensorFlow	-	-	18.20	0.92	1.16	10.58
DeepSpeech	Tensorflow	6.69	5.25	22.46	15.82	3.14	4.41

(a) Results on P600 and K80

		GPU Type and Number							
		M60				GTX 1080Ti			
Network	Framework	1	2	3	4	1	2	4	8
AlexNet	PyTorch	7.53	14.00	6.58	16.73	0.43	1.62	1.15	4.16
	TensorFlow	7.19	6.36	6.91	6.96	4.06	5.36	1.14	1.12
ResNet-50	PyTorch	3.60	20.04	9.58	4.64	12.62	11.93	20.63	4.29
	TensorFlow	2.08	2.79	3.07	21.49	0.68	6.44	1.43	12.06
VGG-19	PyTorch	10.85	18.18	13.81	8.01	24.98	17.40	2.93	14.06
	TensorFlow	7.34	5.06	2.74	6.92	22.88	6.37	24.12	23.56
DeepSpeech	TensorFlow	19.08	12.98	7.09	7.11	5.80	9.63	13.87	5.93

Table 6: MAPE [%] on test set of GPU number extrapolation models

		GPU Type		
Network	Framework	K80	M60	GTX 1080Ti
AlexNet	PyTorch	7.21	14.45	4.98
	TensorFlow	24.75	17.27	8.77
ResNet-50	PyTorch	9.13	9.04	11.76
	TensorFlow	24.58	18.29	6.54
VGG-19	PyTorch	11.78	15.98	24.13
	TensorFlow	8.84	13.52	13.65
DeepSpeech	TensorFlow	11.86	17.49	20.97

training data is used to learn a performance model to predict the execution time of experiments characterized by a feature whose value is larger than all the values of the same feature in the training data.

Table 5 presents the extrapolation accuracy of the performance models trained to evaluate the effect of the batch size parameter. For each performance model trained the largest value of the batch size is twice larger than the largest one included in the training test. This corresponds to predicting the NN learning time on GPUs with an available memory size twice larger than the already characterized devices. Several cases show somewhat large errors, yet none are greater than 28 %.

Tables 6 and 7 present some of the most interesting results. The use of target architectures with the same number and family of GPUs, but with different amounts of memory, is a realistic scenario in a limited number of cases, since generally cloud providers do not allow users to tune the used VMs so deeply. On the contrary, the number of available GPUs is one of the parameters that users can easily control, and one of the most impacting on the overall performance and cloud usage cost. Table 6 reports the accuracy of the models trained for each combination of framework, network, and type of GPU. Since in the available targets the maximum number of K80 and M60

Table 7: MAPE [%] on test set of computational power extrapolation models

Network	Framework	Used Features All	Used Features 5
AlexNet	PyTorch	7.27	27.10
	TensorFlow	5.08	5.08
ResNet-50	PyTorch	18.09	20.74
	TensorFlow	20.23	19.82
Deepspeech	Tensorflow	11.72	10.14

is four, the models are trained by considering experiments with 1, 2, and 3 GPUs in the training set, while the test set includes experiments run on 4 GPUs. On the contrary, since the machine with GTX 1080Ti includes 8 graphic cards, training sets contain experiments run on 1, 2, and 4 GPUs, while 8-GPU experiments are used as test set. Extrapolation on the number of P600 is not possible since the available in-house server 1 includes only two GPUs. In all the scenarios the accuracy is quite good, since the maximum error is 24.75 %, hence the models are able to predict the performance benefit of adding new GPUs of the same type of the existing one within a reasonable error margin. TensorFlow in most scenarios has a larger error, showing how the effects of the implemented interactions between GPUs are more difficult to model compared to PyTorch. Different from the previous types of extrapolations, the results obtained with the GTX 1080Ti are more comparable with the ones of the other types of GPU and in most of the cases they are even better.

Table 7 presents the extrapolation on the computational power feature: in this scenario available data is used to estimate the NN performance on more powerful unseen GPUs. In particular, data about experiments run on P600, K80, and M60 is used to train performance models that are then validated on the data collected on the GTX 1080Ti. It is worth noting that to perform such type of extrapolation, the computational power (P) of the new GPU is not the only necessary information. Indeed, the target architecture is also characterized by disk speed (D): just as highlighted in Section 5.3, there is a strong correlation between P and D, which basically comes from the fact that cloud VM catalogs constrain the possible combinations of GPUs and storage. The value of D can be estimated on the basis of (i) the target available information, or (ii) approximated to the value of the most similar already analyzed target (for example, the disk speed of an Azure VM can be approximated to the disk speed of a VM of the same class for which real data is available), or (iii) retrieved by performing a short profiling run that measures the data load time, as described in Section 3.2. The presented results exploited the third approach. The obtained results are quite good: the MAPE of the performance models is always below 20 %. These results show how using information collected on less powerful GPUs can be effectively used to predict NN training time on better performing hardware without requiring expensive experimental campaigns.

Table 8 lists the first five features selected by the stepwise method when training performance models for this latter scenario investigating computational power extrapolation. The training time of AlexNet implemented on PyTorch heavily depends on the loading time: for this reason, the most significant features are the interaction of disk delay (D), batch size (B), and number of iterations (I). The larger each of these values, the larger the images loading time, consequently the larger the overall training time. On the contrary, in the TensorFlow implementation, the training time, although

Table 8: The most significant features selected in computational power extrapolation models

Network	Framework	Model	MAPE
AlexNet	PyTorch	ID,IB,BD,B/(IPD),1/(IBPD)	27.10
	TensorFlow	ID/P,IBD,B/(PD),IB/D,IBPD	5.08
ResNet-50	PyTorch	IBD,1/B,-IBPD,I/B,BPD/I	20.74
	TensorFlow	IB/P,BP/I,IP,I/BPD,IBPD	19.82
DeepSpeech	TensorFlow	IB,I/B,I/(BP),B/(GP),1/I	10.14

Table 9: MAPE [%] on test set of network depth extrapolation models

			GPU Type		
			M60		
Network	Framework	Max N. IMs	1	2	4
ResNet	PyTorch	4	23.51	27.95	17.40
ResNet	PyTorch	5	24.85	25.11	16.75
ResNet	PyTorch	6	26.76	20.40	16.63
ResNet	PyTorch	8	17.06	7.93	15.99

it still depends on the loading time (in fact, the second selected feature is IBD) also depends on the computational power of the target system. The larger the inverse of the computational power (i.e., $1/P$, which is used in the most significant features), the faster the training process. In the case of ResNet-50 on PyTorch, the first feature (IBD) is still correlated to the loading time, but then there are two correcting factors. The first one ($1/B$) depends on the batch size, while the second ($-IBDP$, which means $IBDP$ has a negative coefficient in the linear regression model) reduces the effect of the first feature on the basis of the computational power. Indeed, even if the computational power P instead of its reciprocal is used, the coefficient of this feature is negative so that the larger P, the larger the reduction of the predicted execution time. The most significant feature selected for predicting performance in case of ResNet-50 implemented with TensorFlow is IB/P. According to this feature, the overall training time is directly proportional to the number of images to be processed and inversely proportional to the computational power of the GPU. It is worth noting that, as also discussed previously, there is a strong correlation between D and P. Finally, the training time of DeepSpeech is not significantly influenced by the computational power nor by the disk speed. For this reason, the most significant features (i.e., IB and I) only depend from the number of iterations I and from batch size B.

The last type of extrapolation analysis refers to a different scenario where the structure of the trained CNN is varied. In particular, the ResNet is modified changing the number of inner modules (IMs) N from 1 to 10 with step 1. The training set is composed of experiments with number of IMs N from 1 to n, while the test set is composed of experiments with number of IMs N from $n+1$ to 10. Performance models are trained and evaluated when n is equal to 4, 5, 6, and 8. Table 9 shows the achieved results. Even in the worst scenario, where the training set is definitely much smaller than the test set (see Appendix), the maximum MAPE is at most about 38 %, since the execution time of the added layers is similar to the execution time of the existing ones, as described in Section 3.1. As expected, the larger the maximum number of IMs in the training set, the better the extrapolation capability of the model, and so its accuracy. Nevertheless, even by limiting the training set to the experiments with up to 5 IMs (hence half of the maximum value, like in the other extrapolation scenarios), the MAPE of the models is at most 25.11 %. In this way, the user can easily evaluate the effect on performance of incrementing the number of IMs in the ResNet. It is worth noting that predicting the effect in terms of improvement of image classification accuracy of the deeper network is out of the scope of this paper.

In all the extrapolation scenarios, despite the small number of available samples (see Appendix), the MAPE on the test set is always below 30 %, allowing the effective use of the proposed models in real scenarios Lazowska et al. (1984).

5.5 Discussion

This work has proposed ML performance models to study the performance of DL networks training time in GPUaaS systems and private clouds. The proposed approach exploits two main sets of features: on one hand, network architecture and hyper-parameters, on the other, hardware characteristics of the target deployment. Such data enable the learning of multiple linear regression models, which, coupled with an established feature selection technique, become accurate prediction tools, with errors below 11 % on average.

Unfortunately, our approach is DL network architecture specific and it is not able to identify a single ML performance model able to account different network architectures characteristics. Generalization on the network architecture can be obtained in case the network has a fixed structure, as in the ResNet, if such assumption does not hold a per-layer approach as, proposed in our previous work Gianniti et al. (2018b), or in Dube et al. (2019) is needed. Indeed, we tried to generalize the applicability of our methodology, by adding new terms to the regression function to account for network characteristics but this was not enough to capture the complex correlations among features and their compound impact on the DL network training time (the obtained errors are above 30%, jeopardizing the use of our performance models in practice).

To achieve good accuracy and generalization on the network architecture, the modeling approach needs to include features governing the computational requirement of a network expressed as flops (as in Gianniti et al. (2018b)) or more high-level features such as the number of weights, activation layers, filters input and output size, etc (as in Dube et al. (2019)). The resulting predictive model will be applicable to arbitrary network architectures but, in that case, are limited to single GPU types.

To actually generalize this approach to predict the training time of NNs composed of whatever type of layer on whatever type of hardware, a large set of data should be collected. This approach would not require to just train a limited set of networks on multiple number and types of GPUaaS, but it would require to generate and train multiple different heterogeneous NNs on all the hardware configurations. The overall result is an exponential growth in terms of the experiments to be performed and consequently an exponential growth in the experimental campaign time and cost.

6 Conclusion

This work has proposed ML performance models able to predict the training time of DL networks. Multiple frameworks, types of NNs and heterogeneous GPU environments have been considered both in GPUaaS scenarios and private clouds.

Differently from previous solutions based on analytical models or other ML-based literature methods, the approach proposed in this paper does not require low-level profiling (e.g., micro-architecture information and/or specific counters) nor simulation of the target applications. Several scenarios of interest have been investigated demonstrating that the trained models, which exploit feature engineering, augmentation, and selection, can be built with limited profiling and can achieve good accuracy and extrapolation capabilities (maximum MAPE equal to about 27 % in the worst case), making them suitable to for adoption in real scenarios.

Future work will exploit the performance models to devise an advanced real-time scheduler able to partition available GPUs among multiple competing users while providing a priori DL networks training time upper bounds. Moreover, the training of DL jobs requiring multiple machines or more than 8 GPUs possibly in disaggregated hardware scenarios (where GPUs are accessed from computing nodes through a fast network) will be also considered. Finally, the models will be also embedded in AutoML-like frameworks where the NN architecture search procedure will be also guided by the target network performance and not only by the model classification accuracy.

Acknowledgments

The results of this work have been partially funded by ATMOSPHERE (grant agreement no. 777154), a Research and Innovation Action funded by the European Commission under the Cooperation Programme, Horizon 2020 and the Ministério de Ciência, Tecnologia e Inovação, RNP/Brazil.

Danilo Ardagna's work is also supported by the AI-SPRINT (grant agreement no. 101016577) H2020 project.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow IJ, Harp A, Irving G, Isard M, Jia Y, Józefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray DG, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker PA, Vanhoucke V, Vasudevan V, Viégas FB, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2016) TensorFlow: Large-scale machine learning on heterogeneous distributed systems. 1603.04467

Amazon (2018a) Amazon EC2 elastic GPUs. URL https://aws.amazon.com/ec2/elastic-gpus/

Amazon (2018b) Amazon Rekognition. URL https://aws.amazon.com/rekognition/

Amodei D, Anubhai R, Battenberg E, Case C, Casper J, Catanzaro B, Chen J, Chrzanowski M, Coates A, Diamos G, Elsen E, Engel J, Fan L, Fougner C, Han T, Hannun AY, Jun B, LeGresley P, Lin L, Narang S, Ng AY, Ozair S, Prenger R, Raiman J, Satheesh S, Seetapun D, Sengupta S, Wang Y, Wang Z, Wang C, Xiao B, Yogatama D, Zhan J, Zhu Z (2015) Deep speech 2: End-to-end speech recognition in english and mandarin. CoRR abs/1512.02595, URL http://arxiv.org/abs/1512.02595

Baghsorkhi SS, Delahaye M, Patel SJ, Gropp WD, Hwu WmW (2010) An adaptive performance modeling tool for gpu architectures. In: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, New York, NY, USA, PPoPP '10, pp 105–114, DOI 10.1145/1693453.1693470, URL http://doi.acm.org/10.1145/1693453.1693470

Bahrampour S, Ramakrishnan N, Schott L, Shah M (2015) Comparative study of deep learning software frameworks. 1511.06435

den Bakker I (2017) Battle of the Deep Learning frameworks: 2017, even more frameworks and interfaces. URL https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

Barnes BJ, Rountree B, Lowenthal DK, Reeves J, de Supinski B, Schulz M (2008) A regression-based approach to scalability prediction. In: Proceedings of the 22Nd Annual International Conference on Supercomputing, ACM, New York, NY, USA, ICS '08, pp 368–377, DOI 10.1145/1375527.1375580, URL http://doi.acm.org/10.1145/1375527.1375580

Bitirgen R, Ipek E, Martinez JF (2008) Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In: Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, Washington, DC, USA, MICRO 41, pp 318–329, DOI 10.1109/MICRO.2008.4771801, URL https://doi.org/10.1109/MICRO.2008.4771801

Caffe2 (2018) A new lightweight, modular, and scalable deep learning framework. URL https://caffe2.ai/

Csurka G (2017) Domain adaptation for visual applications: A comprehensive survey. 1702.05374

Dao TT, Kim J, Seo S, Egger B, Lee J (2015) A performance model for GPUs with caches. IEEE TPDS 26(7):1800–1813

Deng J, Dong W, Socher R, Li L, Kai Li, Li Fei-Fei (2009) Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, pp 248–255, DOI 10.1109/CVPR.2009.5206848

Diamos GF, Kerr A, Yalamanchili S, Clark N (2010) Ocelot: A dynamic optimization framework for bulk-synchronous applications in heterogeneous systems. In: 19th Int'l Conf. Parallel Architecture and Compilation Techniques (PACT 10), IEEE, pp 353–364, DOI 10.1145/1854273.1854318

Draper NR, Smith H (1966) Applied Regression Analysis. Wiley

Dube P, Suk T, Wang C (2019) AI gauge: Runtime estimation for deep learning in the cloud. In: 31st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2019, Campo Grande, Brazil, October 15-18, 2019, IEEE, pp 160–167, DOI 10.1109/SBAC-PAD.2019.00035, URL https://doi.org/10.1109/SBAC-PAD.2019.00035

Foundation M (2019a) Common voice. URL https://voice.mozilla.org/

Foundation M (2019b) Project deepspeech. URL https://github.com/mozilla/DeepSpeech

Gianniti E, Ciavotta M, Ardagna D (2018a) Optimizing quality-aware big data applications in the cloud. IEEE Transactions on Cloud Computing DOI 10.1109/TCC.2018.2874944

Gianniti E, Zhang L, Ardagna D (2018b) Performance prediction of GPU-based deep learning applications. In: 30th Int'l Symp. Computer Architecture and High Performance Computing (SBAC-PAD 18)

Gianniti E, Zhang L, Ardagna D (2019) Performance prediction of gpu-based deep learning applications. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science, CLOSER 2019, Heraklion, Crete, Greece, May 2-4, 2019, SciTePress, pp 279–286

Global Market Insights (2019) GPU as a service market size by product. URL `www.gminsights.com/industry-analysis/gpu-as-a-service-market`

Google (2018a) Cloud AutoML. URL `https://cloud.google.com/automl/`

Google (2018b) GPUs on Compute Engine. URL `https://cloud.google.com/compute/docs/gpus/`

Gupta U, Babu M, Ayoub R, Kishinevsky M, Paterna F, Gumussoy S, Ogras UY (2018) An misc learning methodology for performance modeling of graphics processors. IEEE Transactions on Computer DOI 10.1109/TC.2018.2840710

Hadjis S, Zhang C, Mitliagkas I, Ré C (2016) Omnivore: An optimizer for multi-device deep learning on CPUs and GPUs. `1606.04487`

Hannun AY, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng AY (2014) Deep speech: Scaling up end-to-end speech recognition. CoRR abs/1412.5567

He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. `1512.03385`

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Conf. Computer Vision and Pattern Recognition (CVPR 16), IEEE, DOI 10.1109/CVPR.2016.90

Hong S, Kim H (2009) An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In: Proc. 36th Ann. Int'l Symp. Computer Architecture (ISCA 09), vol 37, pp 152–163

Jia W, Shaw KA, Martonosi M (2012) Stargazer: Automated regression-based GPU design space exploration. In: Int'l Symp. Performance Analysis of Systems & Software (ISPASS 12), IEEE, DOI 10.1109/ISPASS.2012.6189201

Jun TJ, Kang D, Kim D, Kim D (2018) GPU enabled serverless computing framework. In: 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2018, Cambridge, United Kingdom, March 21-23, 2018, pp 533–540

Kerr A, Diamos G, Yalamanchili S (2010) Modeling GPU-CPU workloads and systems. In: Proc. 3rd Workshop General-Purpose Computation on Graphics Processing Units (GPGPU-3), ACM, DOI 10.1145/1735688.1735696

Khomenko V, Shyshkov O, Radyvonenko O, Bokhan K (2017) Accelerating recurrent neural network training using sequence bucketing and multi-gpu data parallelization. CoRR abs/1708.05604, URL `http://arxiv.org/abs/1708.05604`

Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. In: Proc. 25th Int'l Conf. Neural Information Processing Systems (NIPS 12), vol 1, pp 1097–1105

Lazowska ED, Zahorjan J, Graham GS, Sevcik KC (1984) Quantitative System Performance. Prentice-Hall

Lee BC, Brooks DM, de Supinski BR, Schulz M, Singh K, McKee SA (2007) Methods of inference and learning for performance modeling of parallel applications. In: Proc. 12th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP 07), DOI 10.1145/1229428.1229479

Lin M, Chen Q, Yan S (2013) Network in network. `1312.4400`

Liu W, Muller-Wittig W, Schmidt B (2007) Performance predictions for general-purpose computation on GPUs. In: Int'l Conf. Parallel Processing (ICPP 07), IEEE, DOI 10.1109/ICPP.2007.67

Lu Z, Rallapalli S, Chan K, La Porta T (2017) Modeling the resource requirements of convolutional neural networks on mobile devices. In: Proc. Conf. Multimedia (MM 17), ACM, DOI 10.1145/3123266.3123389

Luk CK, Hong S, Kim H (2009) Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO 09), pp 45–55

Maas AL, Hannun AY, Jurafsky D, Ng AY (2014) First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. CoRR abs/1408.2873, URL `http://arxiv.org/abs/1408.2873`

Madougou S, Varbanescu A, de Laat C, van Nieuwpoort R (2016) The landscape of GPGPU performance modeling tools. J Parallel Computing 56:18–33

Microsoft (2018) GPU optimized virtual machine sizes. URL `https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-gpu`

Morais R (2019) A Journey to <10% Word Error Rate. URL `https://voice.mozilla.org/`

Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in PyTorch. In: 31st Conf. Neural Information Processing Systems (NIPS 17)

Peng Y, Bao Y, Chen Y, Wu C, Guo C (2018) Optimus: an efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018, ACM, pp 3:1–3:14

PyTorch (2018) Tensors and dynamic neural networks in Python with strong GPU acceleration. URL `https://pytorch.org`

Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. `1409.1556`

Song S, Su C, Rountree B, Cameron KW (2013) A simplified and accurate model of power-performance efficiency on emergent GPU architectures. In: 27th Int'l Symp. Parallel and Distributed Processing (IPDPS 13), IEEE, DOI 10.1109/IPDPS.2013.73

Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. J Machine Learning Research 15(1):1929–1958

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Conf. Computer Vision and Pattern Recognition (CVPR 15), IEEE, DOI 10.1109/CVPR.2015.7298594

TensorFlow (2018) An open source machine learning framework for everyone. URL `www.tensorflow.org`

Torch (2018) A scientific computing framework for LuaJIT. URL `http://torch.ch`

Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P (2010) Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J Machine Learning Research 11:3371–3408

Zhang Y, Owens JD (2011) A quantitative performance analysis model for GPU architectures. In: 17th Int'l Symp. High Performance Computer Architecture (HPCA 11), IEEE, DOI 10.1109/HPCA.2011.5749745

A Number of Samples in Training Data Used in Performance Prediction Models

This appendix presents the number of samples that have been used to train each of the models presented in Section 5. Table 10 reports the data about the hold-out scenario, while the remaining tables details the size of the training dataset for extrapolation experiments. For example, Table 11 shows how the number of samples of the training set of the extrapolation experiment on the batch AlexNet implemented on PyTorch considering 1 and 2 P600 is 72 and 204 respectively.

Table 10: Number of samples in training data used in interpolation models

Network	Framework	Training set size
AlexNet	PyTorch	3951
	TensorFlow	2326
ResNet-50	PyTorch	1817
	TensorFlow	1782
VGG-19	PyTorch	1312
	TensorFlow	1219
DeepSpeech	TensorFlow	2986

Table 11: Number of samples in training data used in batch size extrapolation models

		GPU Type and Number					
		P600		K80			
Network	Framework	1	2	1	2	3	4
AlexNet	PyTorch	72	204	24	20	24	24
	TensorFlow	48	48	24	24	16	24
ResNet-50	PyTorch	28	44	24	24	24	24
	TensorFlow	28	44	48	24	24	16
VGG-19	PyTorch	-	-	16	64	48	24
	TensorFlow	-	-	16	32	24	24
DeepSpeech	Tensorflow	136	145	240	283	317	316

		GPU Type and Number							
		M60				GTX 1080Ti			
Network	Framework	1	2	3	4	1	2	4	8
AlexNet	PyTorch	39	21	36	21	11	16	17	47
	TensorFlow	18	36	18	18	12	12	12	12
ResNet-50	PyTorch	12	24	48	72	24	24	24	24
	TensorFlow	48	72	48	72	12	12	12	12
VGG-19	PyTorch	20	32	24	29	24	24	24	24
	TensorFlow	20	30	24	28	12	12	12	12
DeepSpeech	TensorFlow	240	281	283	317	52	72	94	475

Table 12: Number of samples in training data used in GPU number extrapolation models

		GPU Type		
Network	Framework	K80	M60	GTX 1080Ti
AlexNet	PyTorch	384	543	947
	TensorFlow	288	744	231
ResNet-50	PyTorch	192	96	96
	TensorFlow	312	768	144
VGG-19	PyTorch	216	144	84
	TensorFlow	288	288	96
DeepSpeech	TensorFlow	768	768	168

Table 13: Number of samples in training data used in computational power extrapolation models

Network	Framework	Training set size
AlexNet	PyTorch	58
	TensorFlow	52
ResNet-50	PyTorch	65
	TensorFlow	52
DeepSpeech	Tensorflow	822

Table 14: Number of samples in training data used in Network depth extrapolation models

			GPU Type and Number		
			M60		
Network	Framework	Max N	1	2	4
ResNet	PyTorch	4	680	131	181
ResNet	PyTorch	5	935	182	233
ResNet	PyTorch	6	1200	240	289
ResNet	PyTorch	8	1757	360	407