IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 16, 2018, accepted September 20, 2018, date of publication October 8, 2018, date of current version November 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2874767

Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications

TIAGO CARNEIRO[1], RAUL VICTOR MEDEIROS DA NÓBREGA[1], THIAGO NEPOMUCENO[2], GUI-BIN BIAN[3], (Member, IEEE), VICTOR HUGO C. DE ALBUQUERQUE[4], (Member, IEEE), AND PEDRO PEDROSA REBOUÇAS FILHO[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará, Fortaleza-CE 60040-531, Brazil
[2]Fraunhofer-Arbeitsgruppe für Supply Chain Services SCS, 90411 Nürnberg, Germany
[3]State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[4]Programa de Pós-Graduação em Informática Aplicada, Universidade de Fortaleza, Fortaleza-CE 60811-905, Brazil

Corresponding author: Gui-Bin Bian (guibin.bian@ia.ac.cn)

This work was supported by the Youth Innovation Promotion Association of the Chinese Academy of Sciences under Grant 218165.

ABSTRACT Google Colaboratory (also known as Colab) is a cloud service based on Jupyter Notebooks for disseminating machine learning education and research. It provides a runtime fully configured for deep learning and free-of-charge access to a robust GPU. This paper presents a detailed analysis of Colaboratory regarding hardware resources, performance, and limitations. This analysis is performed through the use of Colaboratory for accelerating deep learning for computer vision and other GPU-centric applications. The chosen test-cases are a parallel tree-based combinatorial search and two computer vision applications: object detection/classification and object localization/segmentation. The hardware under the accelerated runtime is compared with a mainstream workstation and a robust Linux server equipped with 20 physical cores. Results show that the performance reached using this cloud service is equivalent to the performance of the dedicated testbeds, given similar resources. Thus, this service can be effectively exploited to accelerate not only deep learning but also other classes of GPU-centric applications. For instance, it is faster to train a CNN on Colaboratory's accelerated runtime than using 20 physical cores of a Linux server. The performance of the GPU made available by Colaboratory may be enough for several profiles of researchers and students. However, these free-of-charge hardware resources are far from enough to solve demanding real-world problems and are not scalable. The most significant limitation found is the lack of CPU cores. Finally, several strengths and limitations of this cloud service are discussed, which might be useful for helping potential users.

INDEX TERMS Deep learning, Colab, convolutional neural networks, Google colaboratory, GPU computing.

I. INTRODUCTION

Deep learning applications are present in different aspects of our daily lives, such as web search engines, social network recommendations, natural language recognition, and e-commerce suggestions [1]. This class of application usually rely on heavy computations on massive datasets. Therefore, parallel computing is traditionally considered to run the training process in a feasible time. Graphics processing units (GPU) are massively parallel devices candidates to perform such a parallel task. This kind of accelerator is ubiquitous, accessible, and deliver a high GFlops/Dollar rate [2]. Additionally, the main deep learning frameworks are programmed for NVIDIA GPUs [3].

Hardware resources evolve risks [4]: under and overutilization, depreciation of the hardware, and failures. There are also costs related to maintenance, energy, and human resources. In a research group reality, it may be difficult to keep a robust computer with several GPUs for tests. Furthermore, it is costly to provide for each member of the team a workstation equipped with a high-end GPU.

Nowadays, cloud solutions are attractive because they provide hardware on the fly, and remove the need for maintaining and configuring hardware resources. Cloud platforms such as Amazon, Intel, Azure, and Google Cloud provide in a pay-by-hour manner GPUs and a runtime fully configured for deep learning. Also, NVIDIA offers standalone dockers with pre-configured CUDA environment for deep learning that can be applied to several cloud platforms [5].

Under the scope presented above, Google has created Colaboratory (a.k.a. Colab), a cloud service for disseminating machine learning education and research [6]. The runtime provided by this cloud service is fully configured with the leading artificial intelligence (AI) libraries and also offers a robust GPU. This Google service is linked to a Google Drive account, and it is free-of-charge.

The primary objective of this paper is to study the feasibility of Colaboratory for accelerating deep learning applications. To the best of our knowledge, the present paper is the first work to analyze both performance and resources of Colaboratory, as well as the use of this cloud-based service as a tool for accelerating deep learning applications. To accomplish the primary objective, we performed preliminary experiments and implemented two deep learning applications for computer vision: object classification and object localization and segmentation.

The main contributions of the present research work are the following. We show that Google Colaboratory can be effectively used to accelerate not only deep learning but also other classes of GPU-centric scientific applications. Using Colaboratory's accelerated runtime for training a CNN can be faster than 20 physical cores of a Linux server. Moreover, we provide a detailed analysis of this cloud service regarding hardware sources, performance, and possible applications. Finally, we outline several strengths and limitations of Google Colaboratory, which might be useful for helping potential users.

The remainder of this paper is organized as follows. Section II presents the background topics and related works. Section III brings a preliminary evaluation of Colaboratory's hardware resources and performance. In turn, two computer vision use cases are explored in Section IV: object classification, and object localization and segmentation. An availability experiment is performed in Section V. An availability experiment is performed in Section V. Next, Section VI brings a discussion about the findings of Sections III–V. Finally, Section VII outlines the conclusion of the present research work.

II. BACKGROUND AND RELATED WORKS

This section presents background information and provides an overview of related contributions in the literature that investigate the viability of cloud services for processing high-performance computing applications. The remainder of this section is organized as follows. Section II-A briefly introduces the topic of Deep Learning for Computer Vision. In turn, the Google Colaboratory cloud platform is introduced in Section II-B. Finally, Section II-C brings the related works.

A. DEEP LEARNING FOR COMPUTER VISION

Nowadays, digital image processing is used in a variety of applications, whether for object segmentation into images, extraction of image information or even classification of patterns [7]. Many computer vision applications are intended to use the operating power of deep learning methods, such as Convolutional Neural Networks (CNN) [8].

CNN are mainly applied for analyzing visual imagery, and it has been proven to be crucial in many applications for recognition and decision making [1]. The additional convolutional layers help the network to learn filters that in others traditional algorithms were hand-engineered. The goal of the added layers is to make the network more robust when dealing with transformations in the image. Thus, CNN are also known by space invariant artificial neural networks (SIANN).

Two main problems are encountered when using CNN: specific hardware for good performance and high power consumption of this hardware. For example, health applications need to be recognized for diseases and structures at a high level of accuracy to save lives. It must be done promptly to treat the disease [9]. So CNN's high accuracy index applies to this problem, but high-performance hardware is required to have the life-saving response time. Therefore, GPUs are good candidates for such a task. Besides being massively parallel, this sort of device is also energy efficient [2], [3].

B. GOOGLE COLABORATORY

Before introducing Google Colaboratory, we introduce Jupyter Notebooks, the technology which Colaboratory is based on. Jupyter is an open-source and browser-based tool that integrates interpreted languages, libraries, and tools for visualization [10]. A Jupyter notebook can work either locally or on the cloud. Each document is composed of multiple cells, where each cell contains script language or markdown code, and the output is embedded in the document. Typical outputs include text, tables, charts, and graphics. Using this technology makes easier to share and replicate scientific works since the experiments and results are presented in a self-contained manner [11].

Google Colaboratory (a.k.a Colab) is a project that has the objective of disseminating machine learning education and research [6]. Colaboratory notebooks are based on Jupyter and work as a Google Docs object: can be shared and users can collaborate on the same notebook. Colaboratory provides either Python 2 and 3 runtimes pre-configured with the essential machine learning and artificial intelligence libraries, such as TensorFlow, Matplotlib, and Keras. The virtual machine under the runtime (VM) is deactivated after a period of time, and all user's data and configurations are lost. However, the notebook is preserved, and it is also possible to transfer files from the VM hard disk to the user's Google Drive account. Finally, this Google service provides a GPU-accelerated runtime, also fully configured with the software previously outlined. The Google Colaboratory infrastructure is hosted on the Google Cloud platform.

C. RELATED WORKS

Works that study the viability of cloud services for processing high-performance computing (HPC) applications usually rely on Amazon EC2 service [12]–[14]. The experimental protocol of the listed studies varies. This way, the results and conclusions of the related works are different, even though they use Amazon services. According to Juve *et al.* [12], an instance of Amazon EC2 can achieve a performance comparable to physical systems given similar resources. In contrast, Jackson *et al.* [13] claim that the underlying network infrastructure of the EC2 cloud platform severely limits performance. As a consequence, EC2 instances are much slower than a typical mid-range Linux cluster. In turn, Expósito *et al.* [14] analyze performance bottlenecks in HPC application scalability on the Amazon EC2 service.

Contrasting to the works listed in the last paragraph, Iosup *et al.* [15] compares four different cloud services for processing scientific computing workloads: Amazon EC2, GoGrid, ElasticHosts, and Mosso. According to the results, the evaluated services need an order of magnitude in performance improvement to be useful to the scientific community.

The literature on Google Colaboratory mainly consists of online tutorials based on the official documentation [6]. Among these tutorials, the one by Fuat [16] distinguishes from the others. It gives information about the underlying hardware, presents tutorials on installing well-known artificial intelligence frameworks on Colaboratory, and also provides information about how to access a project on GitHub.

Differently from the work by Fuat, the present research is not a tutorial. Additionally, it is also not a comparison between cloud services. This work analyses different aspects of Google Colaboratory, such as hardware resources, performance, limitations, and possible uses. In this sense, it is similar to Juve *et al.* (2009) and Jackson *et al.* (2010). Moreover, the present research studies the feasibility of Colaboratory for accelerating deep learning and other GPU-centric applications.

III. PRELIMINARY EXPERIMENTS ON GOOGLE COLABORATORY

There is no consensus in the related works whether cloud services are useful for processing compute-intensive scientific applications. Additionally, the related work about Colaboratory brings no performance analysis. This preliminary set of experiments is focused on knowing better Google Colaboratory's hardware resources and finding out what kind of compute-intensive applications this cloud service can be effectively used to accelerate. Furthermore, this section aims at answering the following research question: *Given similar resources, is it possible to run a compute-intensive scientific application on Google Colaboratory and achieve a performance equivalent to the one of dedicated hardware?*

A. METHODOLOGY

In this analysis, a GPU-accelerated backtracking for enumerating all feasible solutions of the N-Queens problem is used as test-case. Backtracking is a problem-solver paradigm that evaluates a solution space in depth-first order. It is present in several research areas, such as artificial intelligence and operations research, and it is considered an essential class of high-performance computing (HPC) application [17]. Moreover, this class of parallel algorithm suits well this evaluation: the GPU portion of the application processes almost 100% of the solution space.

GPU-accelerated backtracking usually consists of two main parts: initial backtracking on CPU, and the search on GPU [18]. The N-Queens problem, the problem of placing N non-attacking queens on a $N \times N$ board, is a classic benchmark for GPU-based backtracking. For the preliminary analysis, the backtracking solves the N-Queens problem for board sizes (N) ranging from 10–18. The solution space ranges from few thousands to several billions of nodes. A serial, multithreaded, and GPU-accelerated versions of the backtracking were implemented.

Two metrics are collected in each experiment: execution time and the rate of node evaluations/second. The execution time is used for calculating the speedup metric, which means the benefit of using parallel programming for solving the N-Queens problem. In turn, the node evaluations/second rate is the performance of the hardware regarding board configurations evaluated/second. The CPU baseline used for calculating the speedup is the serial backtracking executed on one CPU core/thread of Colab testbed, further referred to as T_{serial}. The speedup metric is calculated as follows [17]:

$$\frac{T_{serial}}{T_{parallel}} \tag{1}$$

In the scope of this performance analysis, $T_{parallel}$ means the execution time of the multithread or the GPU-based backtracking.

B. PARAMETERS SETTINGS

All CUDA programs were parallelized using CUDA C 9.0 and compiled with NVCC 9.0 and GCC 5.4. All multithreaded versions were parallelized using OpenMP. The kernel execution time was measured through the `cudaEventRecord` function of CUDA, whereas the overall application time through the `clock` function of C.

Three testbeds were used in the present evaluation: A Linux server, a mainstream workstation, and the hardware configuration under Colaboratory's accelerated runtime, further referred to as *Colab*. All testbeds are summarized in Table 1 and detailed as follows.

- **Server:** Operates under CentOS 7.1 64 bits and it is composed of *two* Intel Xeon E5-2650v3 @ 2.30 GHz with 20 cores, 40 threads, and 32 GB RAM. It is equipped with a NVIDIA Tesla K40m (GK110B chipset), 12 *GB* RAM, 2880 CUDA cores @745 MHz.
- **Mainstream:** Operates under Ubuntu 16.04 LTS 64 bits and it is composed an Intel Core i7 3770, with 4 cores @3.4 GHz, 8 threads, and 8 GB RAM. It is equipped with a NVIDIA GeForce 1050TI (GP107 chipset), 4 GB RAM, 768 CUDA cores @1290 MHz.
- **Colab:** Operates under Ubuntu 17.10 64 bits and it is composed of an Intel Xeon processor (not specified) with two cores @2.3 GHz and 13 GB RAM. It is

TABLE 1. Specification of the three testbeds used in the preliminary performance evaluation: Server, mainstream, and the hardware configuration under colaboratory's accelerated runtime.

Testbed	CPU Clock	CPU Cores/Threads	GPU	CUDA cores	GPU Memory
Server	2.3 GHz	20/40	Tesla K40m	2880	12 GB
Colab	2.3 GHz	$(n/a)/2$	Tesla K80	2496	12 GB
Mainstream	3.4 GHz	4/8	GTX-1050 Ti	768	4 GB

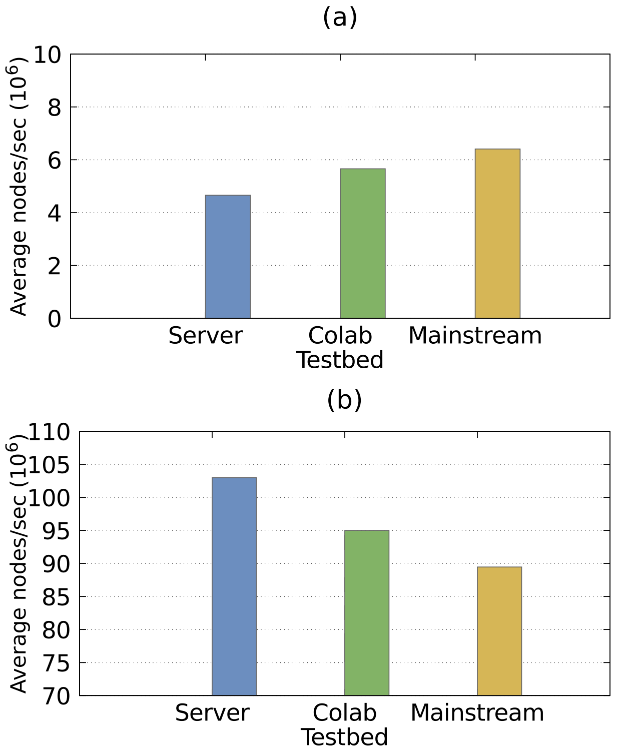

FIGURE 1. The *average* processing rate reached by: (a) *one* core and *one* thread and (b) the GPU of Server, Colab, and Mainstream testbeds enumerating all feasible solutions of the N-Queens. On the graphics, testbed configuration *vs.* average processing rate (in 10^6 nodes/second).

equipped with a NVIDIA Tesla K80 (GK210 chipset), 12 GB RAM, 2496 CUDA cores @560 MHz

C. RESULTS

One can see in Figure 1 the average node evaluations/second rate observed for one core/one thread of each testbed, as well as for the GPU of each testbed. The results are in accordance with Table 1: the highest node evaluations/second rate is observed for the most powerful GPU (K40 - Server). In turn, the free of charge GPU of Colab is superior to the mainstream device (6%) and reaches 93% of Tesla K40's performance. The single core performance is also in agreement with Table 1: the highest performance is observed for the mainstream workstation (6.41×10^6 nodes/sec), which has the CPU with the highest clock. In turn, Colab's serial performance is slightly superior to the serial performance of the Linux server.

Figure 2 presents for each testbed the average speedup reached by the multithreaded and the GPU implementations compared to the serial baseline (Figure 1 - (a)). The results for the GPU-based implementation are also in accordance with Table 1: the server testbed, which is equipped with a Tesla K40 GPU, reaches an *average* speedup of 19.11×. In turn, an average speedup of 18.31× and 17× observed for Colab and Mainstream testbeds, respectively. Moreover, it can be observed in Figure 2 that Colab provides only *one* CPU core that supports *two* threads: no speedup is observed for the multicore implementation running on Colab. In turn, average speedups of 13.9× and 4.27× are observed for the multicore implementation running on Server and Mainstream, respectively.

The results of this section are following Juve *et al.* [12] and give an affirmative answer to the question posed at the beginning of the present section: it is possible to run a compute-intensive scientific application on Google Colaboratory and achieve a performance equivalent to the one of dedicated hardware, given similar resources. In this case, the resources are a GPU and a single CPU core/thread. Furthermore, results show that Google Colaboratory can be effectively used to accelerate GPU-centric applications.

The most significant limitation found concerning the hardware resources is the lack of CPU cores: only one CPU core that supports two threads is provided. In contrast, nowadays ordinary workstations are equipped with multicore CPUs. Therefore, it is not worth, in terms of performance, using Colaboratory for running CPU-based multithreaded applications. However, the lack of CPU cores is not a concern in the scope of deep learning applications. As stated in Section I, the essential deep learning frameworks supports NVIDIA devices.

IV. USING COLABORATORY FOR ACCELERATING DEEP LEARNING APPLICATIONS

This section analyses Google Colaboratory as a tool for accelerating modern and complex deep learning applications. Two computer vision use cases are explored in this evaluation: object detection/classification and object localization/segmentation. This section is organized as follows. Initially, each use case is introduced. Then, the implementation of each method is detailed following the organization of the provided notebook. Next, the methodology of evaluation is presented. Finally, the results are analyzed.

A. OBJECT CLASSIFICATION APPLICATION

In Computer Vision, object classification or detection is the task of assigning an image one label from a set of predefined

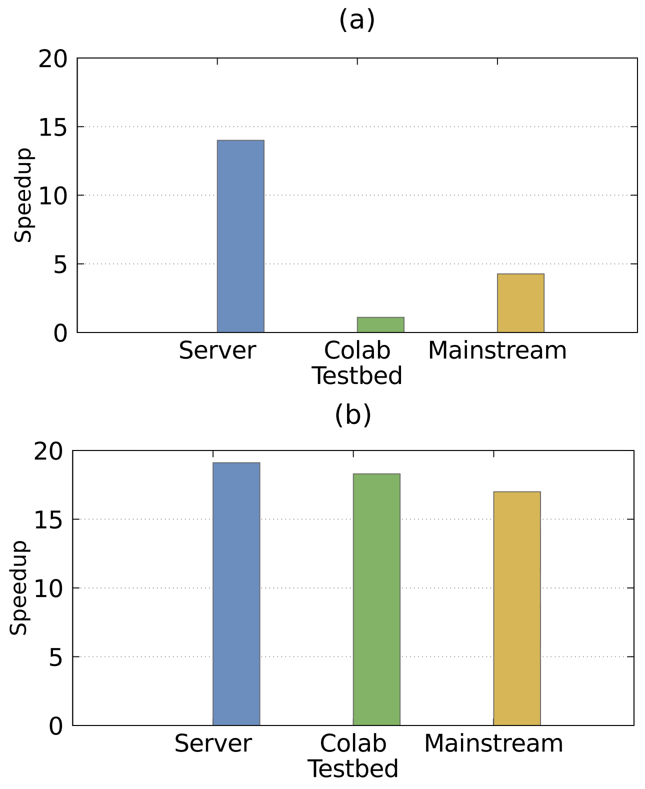

FIGURE 2. **The *average* speedup reached by: (a) *all* cores/threads and (b) the GPU of Server, Colab, and Mainstream testbeds compared to *one* core - *one* thread of Colab's CPU. On the graphics, testbed configuration *vs.* average speedup.**

classes. This task is one of the core problems in Computer Vision and has several practical applications, ranging from lung nodule malignancy classification [19] to the localization of mobile robots [20].

1) IMPLEMENTATION

We implemented the object classification application using the browse-based notebook provided by Colaboratory.[1] The explored object detection implementation is composed of 2 stages. In the first one, the target dataset is loaded into memory and then pre-processed. In this preprocessing phase, the images are normalized between 0 and 1, and their labels are converted to the *one-hot* vector format, which is a binary vector where only one element has its value equals to 1.

In the second stage, a Convolutional Neural Network (CNN) is built, compiled and trained. This network can be outlined as a sequence of 2 *convolution* layers, a *Max-Pooling* layer, a *Dropout* layer, a *flatten* layer, and 2 *dense* layers. In the convolution layers 32 and 64 (3×3) filters were applied using the *ReLU* activation function. In the Dropout layer, each input has a 0.5 probability to be set to zero, reducing the overfitting of the network. In the MaxPooling layer, the subsampling process is performed by returning the max value of a 2×2 moving window, using stride 2. In the flattening layer, the MaxPooling layer output is reshaped into a one-dimensional vector. In the first dense layer there are 128 neurons using the *ReLU* activation function. In the second dense layer there are 10 neurons using the *Softmax* activation function.

Further, this CNN is trained using mini-batch size of 128, a categorical cross-entropy loss and *ADADELTA* optimizer through 12 epochs. Moreover, the images used to train and test the CNN are from the MNIST dataset [8]. This dataset is composed of 70, 000 labeled 28×28 pixel grayscale images of handwritten digits. More specifically, 10, 000 of these images were used as the testing set and the others as the training set.

B. OBJECT LOCALIZATION AND SEGMENTATION APPLICATION

With the advent of facial detection, autonomous vehicles, computer-aided diagnosis and many other systems, the demand for faster and accurate object detection methods has significantly increased. To accomplish such challenging tasks, these methods have not only to classify every object in a given image, but also localize, or even segment them, which substantially raises the task complexity. Fortunately, the most successful approaches to solve these tasks, such as Mask Region-based Convolutional Neural Networks (Mask R-CNN) [21], are available on the Internet.

1) IMPLEMENTATION

The implementation of object localization and segmentation uses Python 3, Keras, and the TensorFlow implementation of Mask R-CNN.[2] This implementation consists of 6 steps. First of all, it is required to download and compile the MASK R-CNN implementation, as well as to download and compile MS-COCO dataset Python API. Next, we change the MASK R-CNN default execution mode configurations to perform only one inference at a time. With the previously installed MS-COCO Python API, we download the MASK R-CNN pre-trained weights on MS-COCO dataset. Then, we build the MASK R-CNN model and then load pre-trained weights into it. Finally, we load the target image into memory and then feed it to the MASK R-CNN model.

C. METHODOLOGY

The objective of the present computational evaluation is to verify whether it is advantageous using Colaboratory for processing modern deep learning applications rather than dedicated hardware. The testbeds introduced in Section III are also used: *Colab*, *Server*, and *Mainstream*. For this evaluation, only the multithreaded and GPU-versions of the deep learning applications previously detailed are considered (Sections IV-A and IV-B).

To compare the testbeds, we extend the speedup to define two metrics: $speedup^t_{gpu}$ and $speedup^t_s$. The first one means

[1]The description of the implementation follows the cells of the notebook, which is available at: https://goo.gl/4r6pZ6.

[2]The description of the implementation follows the cells of the notebook, which is available at: https://goo.gl/CvD6HQ.

TABLE 2. Average time (in seconds) required by the multithreaded and GPU versions of the object classification application for training the CNN on colab and mainstream testbeds. In angled brackets < (*a*), (*b*) >, it is shown: the average speedup observed for the GPU-accelerated implementation compared to: (*a*) - the multithreaded implementation executed on the same testbed ($speedup^t_{gpu}$); (*b*) - the multithreaded implementation executed on 20 cores of the Server testbed ($speedup^t_S$).

Colab			Mainstream		
Multicore	GPU	*Speedup*	Multicore	GPU	*Speedup*
1925 s	135 s	$< 14.77\times,\ 2.93\times >$	1600 s	110 s	$< 14.54\times,\ 3.6\times >$

TABLE 3. Average time (in seconds) required by the multithreaded and GPU versions of the object localization and segmentation application for processing the three images on Colab and Mainstream testbeds. In angled brackets < (*a*), (*b*) >, it is shown: the average speedup observed for the GPU-accelerated implementation compared to: (*a*) - the multithreaded implementation executed on the same testbed ($speedup^t_{gpu}$); (*b*) - the multithreaded implementation executed on 20 cores of the Server testbed ($speedup^t_S$).

Images		Colab			Mainstream		
Name	Resolution	multithreaded	GPU	*Speedup*	multithreaded	GPU	*Speedup*
Highway	(1152, 1600, 3)	29.7 s	2.5 s	$< 11.8\times , 1.3\times >$	9.7 s	2.6 s	$< 3.7\times , 1.2\times >$
Savanna	(686, 1024, 3)	16.2 s	0.7 s	$< 23.1\times , 2.5\times >$	7.7 s	0.6 s	$< 12.8\times , 3.0\times >$
Team	(443, 760, 3)	15.6 s	0.8 s	$< 19.5\times, 2.3\times >$	9.0 s	0.7 s	$< 12.8\times , 2.7\times >$

how much faster it is on average using the GPU of testbed t instead of using all its CPU cores, and it is defined as follows:

$$speedup^t_{gpu} = \frac{T^t_{multicore}}{T^t_{gpu}} \tag{2}$$

Where $T^t_{multicore}$ is the average execution time of the multithreaded version on testbed t using all its CPU cores, and T^t_{gpu} is the same, but executing the application on the GPU of testbed t. In turn, $speedup^t_s$ is how much faster it is on average using the GPU of testbed t than all 20 cores of the Linux server testbed. This metric is defined as follows:

$$speedup^t_s = \frac{T^{server}_{multicore}}{T^t_{gpu}} \tag{3}$$

Where $T^{server}_{multicore}$ is the average execution time of the multithreaded version executed on all 20 physical cores of the Server testbed.

Concerning the first implementation, the average execution time is the one necessary for training the CNN architecture (Section IV-A). The training procedure is repeated 30 times and the average is considered for calculating the metrics given above.

The second implementation performs object localization and segmentation on three arbitrary images from the Internet: *Highway*, *Savanna*, and *Team*. In this second application, the evaluation metric is based on the execution time of the R-CNN model introduced in Section IV-B. More specifically, for each image, the object localization and segmentation is repeated 30 times. Then the average execution time is computed, generating three final results.

It is worth noting that the application execution time was measured using the `time` python module.

D. RESULTS

Concerning the object classification, one can see in Table 2 the average time required by both Colab and Mainstream to train the CNN. This table also brings the average speedup observed for the GPU-accelerate implementation compared to its multithreaded counterpart running on the same testbed ($speedup^t_{gpu}$) and compared to its multithreaded counterpart executed on all cores of the Server testbed ($speedup^t_s$). First of all, it is important to point out that the multithreaded implementation of TensorFlow does not exploit vectorization instructions such as AVX2 and FMA, which is detrimental to the performance of the Server testbed. Using Collaboratory's accelerated runtime to train the CNN is on average 2.93× faster than using all physical cores of the Linux server. In turn, the mainstream workstation is faster than Colab to train the CNN, on both hardware configurations: it is on average 3.6× faster than the Linux server.

It is shown in Table 3 the time required by Colab and Mainstream to perform the object localization and segmentation, as well as the speedup reached by the GPU versions compared to its multithreaded counterpart. The results for the object localization and segmentation follow the ones of the previous experiment: it is more advantageous to perform object localization and segmentation on Colaboratory's accelerated runtime than on 20 physical cores of the Server testbed. The Speedups observed for Colab compared to the Server testbed range from 1.3× to 2.5× ($speedup^t_s$). Moreover, Mainstream's GPU is also slightly superior to Colab's one for performing object localization and segmentation: speedups compared to the Server testbed range from 1.2× to 3.0×.

According to both set of experiments, it is worth investing on GPUs for accelerating deep learning applications. A mid-end GPU, such as NVIDIA GeForce 1050TI can be more than 20× faster than a quad-core CPU, and more than 3× faster than *two* robust CPUs (refer to Tables 2 and 3). Therefore, we conclude that Colaboratory is useful for accelerating complex deep learning applications, *in case the underlying deep learning software supports NVIDIA GPUs*. On the one hand, it is worth using this free-of-charge Google service, especially when the user has no access to at least a mid-end GPU. In this situation, a substantial benefit is observed when exploiting the accelerated runtime. On the other hand, if the underlying deep learning software is not GPU-ready, it is not worth using Colaboratory for performance, due to its lack of CPU cores (as observed in Section III).

Even in situations where the underlying software is not GPU-ready, Colaboratory may be useful. It offers a runtime configured with several frameworks and libraries, e.g., CUDA, Keras, TensorFlow, and OpenCV. Removing from the programmer the responsibility of setting up software may increase productivity. Concerning the notebooks, the output given is also a document, which makes straightforward sharing research results.

V. AVAILABILITY EXPERIMENTS

The training process of a deep learning application may take a long time. According to Colaboratory's documentation [6], a long GPU utilization can be confused with cryptocurrency mining. As a consequence, the user may be forbidden to access the accelerated runtime. The official documentation gives no information concerning how much time it is possible to use the GPU resources, and the tutorial by Fuat [16] claims that the time limit is 12 hours.

This experiment aims at answering the following research questions:

1) *It is possible to use indefinitely the accelerated runtime provided by Google Colaboratory without penalties?*
2) *In case the user loses the access to the accelerated runtime, how much time does it take to get access to the accelerated runtime again?*

1) METHODOLOGY

To verify whether it is possible to use the accelerated runtime uninterruptedly, a CUDA-C program runs on background until the user gets disconnected. To figure it out whether the cloud forbids the user to access the accelerated runtime, after disconnection, the user reconnects, and the program is launched once more.

The CUDA-C program presented in Listing 1 is the one used in this investigation. This code exploits the fact that kernel launches on GPU are asynchronous concerning the host [22, p. 32]. The kernel executed by the GPU performs an infinite loop on the device (*lines* 1–5). The CPU portion of the code (*lines* 6–13) launches the kernel `loop` on GPU (*line* 7). After launching the kernel, the code on CPU prints the date provided by the system every 60 seconds (*line* 8–11). The result is outputted on browser-based the notebook. This way, it is possible to retrieve the results, even in case the VM is lost and restarted.

2) RESULTS

According to the results, the program of Listing 1 can be executed for 12 hours in the first time. Then, the user was disconnected, and the VM restarted. After reconnecting, the program was uploaded and run once more. In the second time, it was observed that the user was disconnected after 3 hours of GPU utilization. Then, it was not possible to connect to the accelerated runtime for 5 hours, and the platform returned the following message: *"No backend with GPU available"*. Additionally, it is possible to connect to the

```
1   __global__ void loop(){
2       int i = 0;
3       while(true)
4         ++i;
5   }
6   int main (){
7       loop<<<1,1>>>();
8       while(true){
9           system("date");
10          usleep(60000000);
11      }
12      return 0;
13  }
```

Listing 1. CUDA-C program used on the availability experiment.

regular runtime while the user is banned from the accelerated one. The experiment was run for a third and fourth times after the ban of 5 hours, and the results were the same.

VI. DISCUSSION AND MAIN INSIGHTS

This section brings a discussion on the use of Colaboratory for accelerating deep learning applications. The content of this section is based on the results of Sections III and IV and on our experiences using Colaboratory. This discussion is given concerning performance, possible applications, and limitations. Finally, the main insights from the present research work are also outlined.

A. PERFORMANCE

Our findings evidence that the Colaboratory's accelerated runtime is adequate not only for accelerating deep learning but also for processing other GPU-centric applications. Using Colaboratory's accelerated runtime for processing the deep learning applications and the GPU-centric combinatorial search is faster than using 20 physical CPU cores. Therefore, it is worth using the cloud service in question than, for instance, a robust server with no GPU, a laptop, or a workstation that needs to be configured and has a mid-end GPU. On the one hand, the free-of-charge hardware resources provided by Colaboratory are far from enough to solve demanding real-world problems and are not scalable. On the other hand, the main deep learning frameworks are programmed for NVIDIA GPUs, and the performance of the GPU provided by Colaboratory may be enough for several profiles of researchers and students.

In situations where better hardware resources than the ones provided by Colaboratory are available, it is possible to install Jupyter locally and run a Colab notebook. It is also possible to execute the content of each cell individually, without using Jupyter. In this case, it is required to configure the whole software stack: GPU drivers, CUDA toolkit, artificial intelligence libraries, programming languages, and so on. It is worth to point out another aspect of Colab: the Internet infrastructure is fast, especially when accessing resources from other Google services, such as Drive and Storage. Thus, it may be interesting for some users manipulating datasets using Colaboratory rather than a residential Internet connection.

B. OTHER POSSIBLE APPLICATIONS

Besides the use for accelerating deep learning and other GPU-centric applications, Colaboratory may be interesting for teaching HPC. The accelerated runtime is fully operational, which keeps inexperienced users from configuring the CUDA Toolkit, compilers, and GPU drivers. Moreover, a teacher can share notebooks containing lessons and code ready for execution. The free-of-charge GPU is superior to several low-end and mid-end gamer GPUs, which democratizes the access to such an expensive device.

In a scenario of a research group with a low budget, Colaboratory can also be useful. The accelerated runtime can be exploited for accelerating deep learning and other applications, keeping the members from software configuration and hardware maintenance. Moreover, it is worth using the free-of-charge resources of Colaboratory to decide whether to buy a dedicated computer or contract cloud-based services for deploying applications. Finally, Jupyter notebooks are a straightforward way of collaboration between team members.

C. LIMITATIONS

Besides the positive aspects previously discussed, this cloud service also presents limitations that are worth to discuss. First of all, it is difficult to program directly on a Colaboratory notebook. Programs written in compiled languages must be compiled on the user's computer, and then uploaded for execution. Therefore, in case the user has no GPU, it is not straightforward using the accelerated runtime for testing and validating a GPU-based application. Moreover, there are limits on both virtual machine lifetime and GPU utilization. The VM and all files are lost after 12 hours, and the user needs to reconfigure the runtime from scratch. Thanks to the notebooks saved in Google Drive, this reconfiguration is straightforward but may take a while.

There may be situations where it is required to use Google Drive as an interface between Colaboratory and the user's computer. For instance, several deep learning applications receive huge datasets as input. In such a situation, the user must learn the API used by Drive to transfer files from Drive to the VM hard disk. The main limitation concerning this fact is the limit on transfers between Colaboratory and Drive. One way of coping with such a restriction is creating scripts to compress the dataset. Finally, there are no contracts or guarantees, which means that the hardware resources may change along the time, Google may finish the Colaboratory project, and so on. Thus, users may lose computer resources or even means of visualizing saved notebooks.

D. MAIN INSIGHTS

The following summarizes the main insights from our study on Colaboratory as a tool for accelerating deep learning applications:

- Colaboratory can be effectively used to accelerate not only deep learning applications but also other GPU-centric applications.
- The hardware resources provided by Colaboratory are far from enough to solve demanding real-world problems. However, these resources may be enough for several profiles of researchers.
- Colab's performance is equivalent to dedicated hardware, given similar resources.
- The user needs to learn Google API principles to fully exploit Colaboratory features.
- Jupyter Notebooks are a straightforward tool for sharing knowledge.

VII. CONCLUSION AND FUTURE WORKS

This work presented a study about the feasibility of Google Colaboratory for accelerating deep learning for computer vision applications. Results show that Colaboratory hardware resources can reach a performance similar to dedicated hardware. Results also show that it is wort to run experiments on Colaboratory in case the research group has no GPU more robust than a K80. Moreover, it is possible to accelerate other GPU-centric applications than deep learning related ones, with no need for CUDA runtime configuration. Besides the performance, it is interesting using this cloud service because it is straightforward to share notebooks with code and outputs.

The present research also found limitations of Colaboratory regarding deep learning and HPC. It is worth to notice the lifetime of a VM, time limit for GPU utilization, the need of transferring data to/from Google drive or Git, the limit of data transfer between Drive and Colab, and the lack of CPU cores. Furthermore, the hardware provided by Colaboratory is not scalable and far from necessary for solving bigger problems.

As a future research direction, we propose an application that breaks the problem in such a way it could be executed on more than one Colaboratory instance. This way, it would be possible to use more than one GPU without paying for an expensive service. Furthermore, Jupyter notebooks can be executed locally or on other cloud services. We also propose as future research investigating the use of notebooks on other platforms, such as local machines and cloud services. Finally, future investigations of Google Colaboratory as a tool for teaching HPC are also considered, as suggested in the previous section.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[2] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in heterogeneous computing," *Sci. Program.*, vol. 18, no. 1, pp. 1–33, 2010.

[3] NVIDIA Corporation, "TESLA V100 performance guide: Deep learning and HPC applications," White Paper, 2016. [Online]. Available: https://images.nvidia.com/content/pdf/v100-application-performance-guide.pdf

[4] M. Armbrust *et al.*, "Above the clouds: A Berkeley view of cloud computing," Dept. EECS, Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.

[5] NVIDIA Corporation, "Introduction to NVIDIA GPU cloud," Appl. Note DA-08792-001, 2018.

[6] Google. (2018). *Colaboratory: Frequently Asked Questions*. Accessed: Jun. 21, 2018. [Online]. Available: https://research.google.com/colaboratory/faq.html

[7] R. Girshick *et al.*, "Deep learning for computer vision," *Comput. Vis. Image Understand.*, vol. 164, pp. 1–2, 2017. [Online]. Available:

http://www.sciencedirect.com/science/article/pii/S1077314217301972, doi: 10.1016/j.cviu.2017.11.006.
[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[9] P. P. R. Filho, E. D. S. Rebouças, L. B. Marinho, R. M. Sarmento, J. M. R. Tavares, and V. H. C. de Albuquerque, "Analysis of human tissue densities: A new approach to extract features from medical images," *Pattern Recognit. Lett.*, vol. 94, pp. 211–218, Jul. 2017.
[10] F. Pèrez and B. E. Granger, "IPython: A system for interactive scientific computing," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 21–29, May/Jun. 2007.
[11] B. M. Randles, I. V. Pasquetto, M. S. Golshan, and C. L. Borgman, "Using the Jupyter notebook as a tool for open science: An empirical study," in *Proc. ACM/IEEE Joint Conf. Digit. Libraries (JCDL)*, Jun. 2017, pp. 1–2.
[12] G. Juve *et al.*, "Scientific workflow applications on Amazon EC2," in *Proc. 5th IEEE Int. Conf. E-Sci. Workshops*, Dec. 2009, pp. 59–66.
[13] K. R. Jackson *et al.*, "Performance analysis of high performance computing applications on the Amazon Web services cloud," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Nov./Dec. 2010, pp. 159–168.
[14] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "Performance analysis of HPC applications in the cloud," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 218–229, 2013.
[15] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.
[16] Fuat. (2018). *Google Colab Free GPU Tutorial*. Accessed: Jun. 25, 2018. [Online]. Available: https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d
[17] K. Asanović *et al.*, "The landscape of parallel computing research: A view from Berkeley," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2006-183, 2006.
[18] T. Carneiro Pessoa, J. Gmys, F. H. de Carvalho, Jr., N. Melab, and D. Tuyttens, "GPU-accelerated backtracking using CUDA dynamic parallelism," *Concurrency Comput., Pract. Exper.*, vol. 30, no. 9, p. e4374, 2017, doi: 10.1002/cpe.4374.
[19] G. Litjens *et al.*, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.
[20] Z. Chen *et al.*, "Deep learning features at scale for visual place recognition," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May/Jun. 2017, pp. 3223–3230.
[21] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 2980–2988.
[22] *CUDA C Programming Guide (Version 9.1)*, NVIDIA, Santa Clara, CA, USA, 2018.

TIAGO CARNEIRO was born in Fortaleza, Brazil, in 1986. He received the B.Sc. and M.Sc. degrees from the State University of Ceará, Brazil, in 2009 and 2012, respectively, the Ph.D. degree in computer science from the Federal University of Ceará, Brazil, in 2017, all in computer science. He is currently a Post-Doctoral Fellow at the Instituto Federal de Educação, Ciência e Tecnologia do Ceará, Brazil. His research interests include the use of heterogeneous architectures for solving combinatorial optimization problems.

RAUL VICTOR MEDEIROS DA NÓBREGA received the bachelor's and master's degrees in computer science from the Instituto Federal de Educação, Ciência e Tecnologia do Ceará, Brazil, in 2016 and 2018, respectively. He is currently a member of the Laboratory of Image Processing, and Computational Simulation (LAPISCO), Instituto Federal de Educação, Ciência e Tecnologia do Ceará. His research interests include the classification of pulmonary nodules through techniques of digital image processing and artificial intelligence.

THIAGO NEPOMUCENO was born in Fortaleza, Brazil, in 1990. He received the B.Sc. and M.Sc. degrees in computer science from the State University of Ceará, where he developed research on optimization and meta-heuristics, in special genetic algorithms. He is currently pursuing the Ph.D. degree with the University of Erlangen–Nuremberg. He is also a full-time Employee at the Fraunhofer-Arbeitsgruppe für Supply Chain Services SCS, where he researches in the field of Internet of Things.

GUI-BIN BIAN (M'15) received the bachelor's degree in mechanical engineering from the North China University of Technology, Beijing, China, in 2004, and the master's and Ph.D. degrees in mechanical engineering from the Beijing Institute of Technology, Beijing, in 2007 and 2010, respectively. He is currently an Associate Professor with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, where his research interests include design, sensing, and control for medical robotics.

VICTOR HUGO C. DE ALBUQUERQUE received the degree in mechatronics technology with the Federal Center of Technological Education of Ceará in 2006, the M.Sc. degree in teleinformatics engineering from the Federal University of Ceará in 2007, and the Ph.D. degree in mechanical engineering with emphasis on materials from the Federal University of Paraíba in 2010. He is currently an Assistant VI Professor with the Graduate Program in Applied Informatics, Universidade de Fortaleza. He has experience in computer systems, mainly in the research fields of applied computing, intelligent systems, and visualization and interaction, with specific interest in pattern recognition, artificial intelligence, image processing and analysis, and automation with respect to biological signal/image processing, image segmentation, biomedical circuits, and human/brain–machine interaction, including augmented and virtual reality simulation modeling for animals and humans. In addition, he has researched in the microstructural characterization field through the combination of non-destructive techniques with signal/image processing and analysis and pattern recognition.

PEDRO PEDROSA REBOUÇAS FILHO received the degree in mechatronics engineering from the Federal Institute of Ceará, Brazil, in 2008, and the M.Sc. degree in teleinformatics engineering, in the field of biomedical engineering, and the Ph.D. degree in teleinformatics engineering from the Federal University of Ceará, in 2010 and 2013, respectively. He has been an Assistant Professor with the Instituto Federal de Educação, Ciência e Tecnologia do Ceará, since 2008. From 2015 to 2016, he was a Post-Doctoral Researcher at the University of Porto, Portugal. He has co-author over 100 articles in national and international journals and conferences. Also, he has been involved in several research projects in the field of computer vision, medical image, and embedded systems.