Journal of Computer Science

Original Research Paper

Performance Analysis of Deep Learning Libraries: *TensorFlow* and *PyTorch*

[1]Felipe Florencio, [1]Thiago Valença, [1]Edward David Moreno and [2]Methanias Colaço Junior

[1]Departamento de Computação, Universidade Federal de Sergipe, São Cristovão, Brazil
[2]Departamento de Sistema de Informação, Universidade Federal de Sergipe, Itabaiana, Brazil

Article history
Received: 01-01-2019
Revised: 25-02-2019
Accepted: 11-04-2019

Corresponding Author:
Felipe Florencio
Departamento de Computação, Universidade Federal de Sergipe, São Cristovão, Brazil
Email: felipe.florencio@dcomp.ufs.br

Abstract: Through the increase in deep learning study and use, in the last years there was a development of specific libraries for Deep Neural Network (DNN). Each one of these libraries has different performance results and applies different techniques to optimize the implementation of algorithms. Therefore, even though implementing the same algorithm and using different libraries, the performance of their executions may have a considerable variation. For this reason, developers and scientists that work with deep learning need scientific experimental studies that examine the performance of those libraries. Therefore, this paper has the aim of evaluating and comparing these two libraries: *TensorFlow* and *PyTorch*. We have used three parameters: Hardware utilization, hardware temperature and execution time in a context of heterogeneous platforms with CPU and GPU. We used the MNIST database for training and testing the *LeNet* Convolutional Neural Network (CNN). We performed a scientific experiment following the Goal Question Metrics (GQM) methodology, data were validated through statistical tests. After data analysis, we show that *PyTorch* library presented a better performance, even though the *TensorFlow* library presented a greater GPU utilization rate.

Keywords: *Tensorflow*, *PyTorch*, Comparison, Evaluation Performance, Benchmarking, Deep Learning Library

Introduction

Deep learning is an artificial intelligence field that allows computers to learn with experience and understand the world in terms of a concept hierarchy, with each concept defined by its relationship with simpler concepts. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, it is called deep learning (Goodfellow *et al*., 2016).

The deep learning idea is to train an Artificial Neural Network (ANN) of multiple layers in a set of data in order to allow it to deal with real world tasks. Although the theoretical concepts behind are not new, deep learning has become to be a trend in the last decade due to many factors, including its well-succeed application in a variety of problem solution (many of them are potentially commercial, such as the development of new computer architectures with a higher level of parallelism, the design of Convolutional Neural Network (CNN) and a higher accessibility to high performance computers (Shatnawi *et al*., 2018; Verhelst and Moons, 2017) (Raschka, 2015).

There are many deep learning libraries, such as *TensorFlow*, *Theano*, *CNTK*, *Caffe*, *Torch*, *Neon* and *PyTorch*. Each one of these libraries has different features of performance and applies different techniques to optimize the implementation of algorithms. Therefore, even though implementing the same algorithm in different structures, the performance of these different implementations may have a considerable variation (Bahrampour *et al*., 2015; Shatnawi *et al*., 2018).

Since there are a variety of open source libraries available, developers and scientists that work with deep learning need scientific experimental studies that point out which library is the most suitable for determined application.

Being that, the present work evaluates and compares the *Tensorflow* library and *PyTorch* library focusing on their hardware utilization, hardware temperature and execution time in a context of heterogeneous platforms with CPU and GPU. We have used the Modified National

Institute of Standards and Technology (MNIST) database for training and testing the *LeNet* CNN.

For the reason that there are a variety of open source libraries available, developers and scientists that work with deep learning need scientific experimental studies that point which library is the most suitable for determined application. Being that, the present work has as aim to evaluate and compare the *Tensorflow* library and *PyTorch* library focusing on their hardware utilization, hardware temperature and execution time in a context of heterogeneous platforms with CPU and GPU. It was used the Modified National Institute of Standards and Technology (MNIST) database for training and testing the *LeNet* CNN.

The novelty that this article presents is the performance evaluation of *PyTorch* library, the use of GPU and CPU utilization rate as evaluation metrics and the use of statistical tests for validating the obtained data during the experiment. As a result, the *PyTorch* library presented a superior performance when compared with *TensorFlow* library, through data analysis, it was verified that during execution using *PyTorch* there is a smaller GPU utilization rate. It is possible to conclude that the communication bottleneck between CPU and GPU is a relevant factor for *TensorFlow* presenting an inferior performance than *PyTorch*, once *TensorFlow* uses more the resources of the GPU.

For a better understanding in how the results were obtained, the present paper is divided into eight sections. In section 2 presents the used method. In Section 3, the conceptual bases are presented, also the used neural network is described and the examined libraries are presented. In Section 4 related works are presented. In section 5, the definition and the experiment planning for making the frameworks comparison is showed. The Section 6 has the execution phases of the experiment, including how the data were collected. In Section 7, the analysis and interpretation of the obtained results are done, besides that, it is shown the threats to validity in this work. Finally, in Section 8 is showed the conclusion and the possible future works.

Method

The paper consists in an experimental study of performance evaluation of two deep learning libraries (*TensorFlow* and *PyTorch*) in a heterogeneous computational system with CPU and GPU. For experimental research we have used as benchmark framework the *LeNet* CNN for training and inferring data in MNIST dataset.

In a prior moment of this research we have selected some related works that benchmark deep learning libraries. The analysis of the related works demonstrated the absence of performance analysis of *PyTorch* library, in the analysis process also was verified that the selected experiments do not take in consideration the utilization rate of hardware components and that, in general, the authors do not use statistical tests for validating the data extracted from the experiment. We have selected the *PyTorch* library, for being little explored and the *TensorFlow* library since it is popular and for serving as performance reference.

We used six (6) metrics: (i) execution time for inference algorithm, (ii) execution time for training algorithm, (iii) GPU utilization rate, (iv) CPU utilization rate, (v) GPU temperature and (vi) CPU temperature. The execution time is used to verify which library presents the best performance and the utilization rate is used to investigate possible causes for this performance. The selected CNN for evaluating performance of the libraries is *LeNet* and the selected dataset is MNIST. The reason for that is the availability of preexisting codes for both libraries.

After the materials selection, it was performed an *in silico* experiment using a heterogeneous computational system. The experiment consisted in preparing the execution environment, adapting the codes extracted from the library's official repository, implementing scrips, executing the codes and extracting data of each execution. The experiment's organization followed the Goal Question Metric (GQM) method as indicated by Basili *et al*. (1994).

After data extraction, it was performed Kolmogov-Smirnov (KS) and *Wilcoxon* statistical tests for validating the data. After validation, the data could be analyzed and evaluated.

Conceptual Bases

This section presents some concepts that are necessaries for understanding this work.

Convolutional Neural Networks

The Convolutional Neural Networks (CNNs) are a kind of biologically inspired feed-forward neural network, it is developed to imitate the behavior of an animal visual cortex (da Costa e Silva Franco, 2016). The basic concept of CNNs goes back to 1979 when Fukushima (1979) proposed an artificial neural network including simple and complex cells that were very similar to the convolutional and pooling layers from modern CNN.

CNNs composed by many non-linear data processing layers, where the output of each inferior layer feed the input of its immediately superior layer (Deng, 2014). They use convolution in place of general matrix multiplication in at least one of their layers (Goodfellow *et al*., 2016). The layers of CNNs may be of three kinds: Convolutional layer, pooling layer and dense layer.

Convolutional Layer

The convolutional layer consists in a set of resource maps, that are generated from a convolutional operation on input data or another resource map. Each convolutional

layer defines an input representation in a determined abstraction level (da Costa e Silva Franco, 2016).

In its general form, the convolution is a linear operator, that from two given functions, results in a third one which is the sum of the product of those functions along the region implied by their superposition in function of the existent displacement between them (Goodfellow *et al*., 2016). It has the following form:

$$\int x(a)w(t-a)da \tag{1}$$

In which $x(t)$ and $w(a)$ are two given functions and $s(t)$ the resulting function. The convolution operation is typically denoted by an asterisk:

$$s(t)=(x*w)(t) \tag{2}$$

In case of convolutional networks, the first argument (in this example, function x) for the convolution is many times called input and the second argument (in this example, function w) is called kernel. The output is commonly called Feature Map (Goodfellow *et al*., 2016).

In Fig. 1 there is an example of 2-D Convolution without flow of kernel. The output is limited to only the position where the kernel is fully inside of the image called valid convolution. In the figure there are arrows indicating as the left superior of the output is formed applying the kernel to the left superior region corresponding to input.

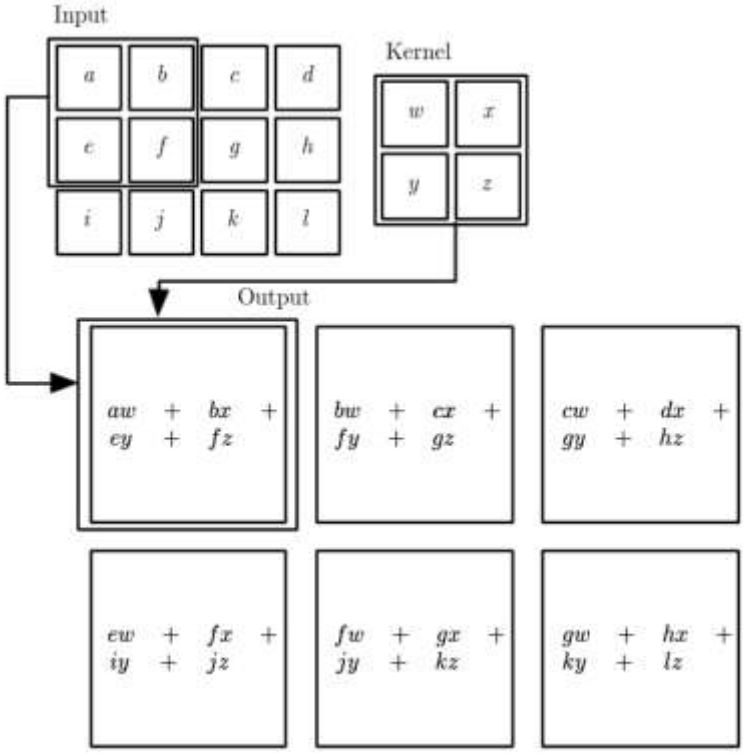

Fig. 1: An example of 2-D convolution (Goodfellow *et al*., 2016)

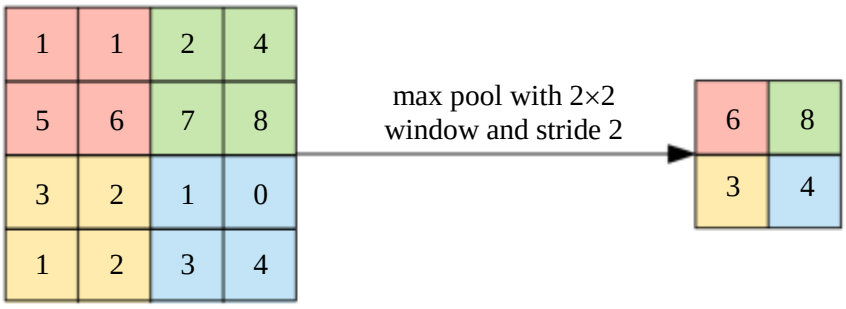

Fig. 2: Example of Max-pooling layer obtained from a features map input (Dertart, 2017)

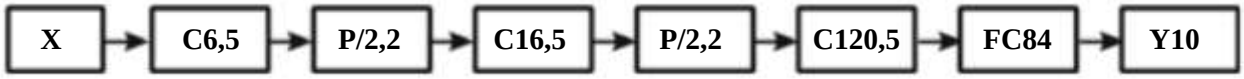

Fig. 3: Representing *LeNet*-5 using DAG (Hamed Habibi Aghdam, 2017)

Pooling Layer

The pooling layer serves to progressively reduce the representation spatial size, reduce the number of parameters and the quantity of computation in the network and, therefore, also control overfitting (GBT, 2018a).

A pooling algorithm commonly used is the Max-pooling, that extracts sub-regions of a resource map (for example, blocks of 2×2 pixels), maintain its maximum value and discard all other values (da Costa e Silva Franco, 2016; GBT, 2018a). This extraction is shown in Fig. 2 in which the Max-pooling layer is obtained from a features map input.

Dense Layer

Fully connected layers, similar to layers of multi-layers Perceptrons (MLP), that perform classification of extracting features by convolutional layers and decreased by pooling layers. In each dense layer, all nodes are connected to all nodes from the previous layer (Raschka, 2015; da Costa e Silva Franco, 2016; GBT, 2018a).

LeNet CNN

Lecun *et al*. (1998) proposed the paradigm of weigh sharing and derived layers from convolution and grouping. Then, they projected a Convolutional Neural Network that is called *LeNet*-5. The architecture of this CNN is illustrated in Fig. 3.

In this Directed Acyclic Graph (DAG), *Ca*, *b* shows a convolutional layer with size filters *b*×*b* and the phrase/*a* in any node shows the last one from this operation. Besides that, *P*/*a*, *b* denotes an operation of pooling with last *a* and size *b*, FCa shows a fully connected layer with the neurons, *Ya* shows an output layer with a neuron. This CNN was originally proposed for recognizing manuscript digits, it consists of four layers of convolution pool. The input of a Neural Network is a 32×32 single channel image. Besides that, the last layer of pooling is connected to the fully connected layer. (Hamed Habibi Aghdam, 2017).

In this work it was used *LeNet* with the backpropagation algorithm for CNN training.

Libraries

In this section the evaluated libraries on this paper are presented.

Tensorflow

Tensorflow (Abadi *et al*., 2015) is an open source library for numeric computation originally developed by the Google Brain Team. It has a flexible architecture that allows easy implantation in different architectures (CPU, GPU and TPU), for this reason it is used in desktops, clusters and mobile devices. It supports machine learning and deep learning been also used in a variety of scientific domains (GBT, 2018b).

Written from another deep learning library called DistBelief, *TensorFlow* is implemented based on directed graphs. In these graphs, nodes represent mathematics, operations and edges represents the flow of data among the nodes, what makes *TensorFlow* used in any domain that the computation can be designed as a flow network (Parvat *et al*., 2017; Shatnawi *et al*., 2018).

TensorFlow is developed as a Python API in C/C++ language seeking to achieve improvements in performance (Parvat *et al*., 2017). It is available for Windows, Linux, Mac OS and on mobile platforms such as AndroidOS and Raspberry (Parvat *et al*., 2017).

PyTorch

Pytorch (Paszke *et al*., 2017) is developed by *PyTorch* Core Team, a group formed by many organizations such as: Nvidia, Facebook Open Source, ParisTech, Twitter, Universite Pierre et Marie Curie, University of Oxford, Stanford University, Uber, among others. The development focus is to produce a framework for tensors and dynamic neural networks in Python with strong acceleration of the GPU.

It is a library built with the aim to be deeply integrated to Python, differently of *TensorFlow* that is a link between Python in a monolithic structure of C/C++. Two of its principal features are: Tensor computation (as Numpy) with strong use of GPU and deep neural networks built in an automatic differentiation system in reverse mode that allows the random way that neural network behaves with overhead (*PyTorch*, 2018a; GBT, 2018b).

PyTorch is integrated with accelerated libraries such as Intel MKL and NVIDIA (CuDNN, NCCL) in order to maximize the performance. At the core, it is CPU and GPU Tensor and neural network back-ends are written as independent libraries with a C99 API. It can be used as a Numpy substitute for better utilization of the GPU power.

Related Works

This work presents a controlled experiment for the evaluation of two deep learning libraries. In this context, this section shows some papers that have a similar approach.

Shi *et al*. (2016) performed a comparative study among many deep learning libraries, including *Caffe*,

MXNet, *CNTK*, *TensorFlow* and *Torch*. The study considers three types of neural networks, including MLP networks, convolutional neural networks (*AlexNet* and *ResNet-50*) and recurrent neural networks (*LSTM-32 e LSTM-64*), being executed in two platforms of CPU and three platforms of GPU. The comparison is done through execution time and convergence rate. It was used sets of synthetic data (the authors did not inform the used dataset) for measuring the performance of execution time and real-world set of data for measuring the convergence rate in their experiments. Hardware temperature was not used.

Goldsborough (2016) presents a comparative study of five deep learning libraries: *Caffe*, *Neon*, *TensorFlow*, *Theano* and *Torch*. The comparison takes into consideration the extensibility, hardware utilization and velocity, using gradient computation time and forward time as metrics. The evaluation was accomplished using CNN in CPUs and GPUs, but it was not taken into account the hardware temperature.

Kruchinin *et al*. (2015) presented a comparative analysis of some popular deep learning libraries and freely available: *Caffe*, *Pylearn2*, *Torch e Theano*. The authors execute a MLP network and a CNN network, whose model is not detailed, for training the *MNIST* dataset. Accuracy and execution time are calculated, as also the usability and flexibility of each library are measured. As a conclusion, *Caffe* and *Torch* libraries were considered the most suitable for training MNIST dataset.

Kovalev *et al*. (2016) shows a comparative study of velocity (training and classification time), classification precision and of implementation complexity (number of code lines) among deep learning libraries: *Theano*, *Torch*, *Caffe*, *TensorFlow* and Deeplearning4j. This study limited itself to the neural network fully connected (MLP) executing in CPU architectures, these networks and CPU architectures are not the most indicated tools for exploring the potential of deep learning libraries. The study also does not explore the impact of hardware temperature and scalability.

Liu *et al*. (2018) presented project considerations, metrics and challenges for the development of an effective benchmark for deep learning softwares and illustrate some observations through a comparative stead of three deep learning libraries: *TensorFlow*, *Caffe* and *Torch*. The experiments consisted in executing a *LeNet* network for learning and inference in MNIST dataset and CIFAR-10, the authors executed the *LeNet* network with two different optimization algorithms: ADAM and SGD. The results illustrated that these libraries are optimized with their standard settings, but the optimized standard setting in a set of specific data may not work effectively for other sets of data with respect to execution time performance and to learning precision.

Shatnawi *et al*. (2018) executed a comparative study among three open source libraries for deep learning: *TensorFlow*, CNTK and *Theano*. The performed evaluation of this work considers the CPU and GPU performance using convolutional neural networks (CNN) and the MNIST dataset and CIFAR-10 measuring processing time according to the number of used threads. The results were the following: Related to the image recognition dataset (MNIST and CIFAR-10) CNTK presented the best performance compared to *TensorFlow* and *Theano* in terms of GPU and CPU multithreading, but in CIFAR-1 processing using 8,16 and 32 threads in CPU *Tensorflow* was faster than CNTK, *Theano* revealed being slower than the other libraries. The authors did not inform the used CNN architectures.

Fonnegra *et al*. (2017) evaluated and compared the following libraries: *TensorFlow*, *Theano* and *Torch*. The comparison is performed through the implementation of recurrent and convolutional architectures for classifying images of two datasets: MNIST and CIFAR-10. The utilized architectures were *LeNet* and LSTM. For evaluating performance, the authors calculated the forward time (execution time of gradient). As a result, concluded that *Torch* requires the shortest computational time for each iteration in CPU and GPU configurations, it means, it reports the less usage of time for training and gradient computation for configurations of CPU and GPU, but also report the greatest time for testing. For processing architectures with LSTM cores, *TensorFlow* was faster than *Theano* in all the cases, except by recognizing task CIFAR-10 time of the test. The evaluation does not take into consideration the utilization and the temperature of the hardware.

None of the found related works present a performance evaluation of *PyTorch* library, they do not take into consideration the CPU and GPU utilization rate and do not apply statistical tests for validating data.

Definition and Planning

In this section, it is showed the planning strategy of the proposed controlled experiment. The next subsections present aim and planning of the experiment (context, dependent and independent variables, hypothesis, object of analysis, experiment project and instrumentation).

Definition of the Aim

The aim of this work is to evaluate (according to execution time, temperature and scalability) the *PyTorch* and *TensorFlow* libraries, verifying its behavior in learning and inference phases of convolutional neural network.

The evaluation was accomplished with an *in silico* experimental study-where the situations will be described by computational models.

Following the aim definition formalization of *GQM* model, proposed by Basili *et al*. (1994), the aim can be rewritten as: Analyse the deep learning libraries *TensorFlow* and *PyTorch*, for the purpose of compare them with respect to execution time, scalability and

hardware temperature from the point of view of researchers, scientists and developers that work with deep learning in the context of *LeNet* convolutional neural network executing in heterogeneous platforms with CPU and GPU.

Planning

Context Selection

The experiment was *in silico* and used the *Dell Inspiron* 15 *Gaming i*15-7567 notebook. For each library it was executed the training algorithm (learning) and the inference for a *LeNet* convolutional neural network (CNN) processing a set of *MNIST* data, what allowed to verify the behavior of the libraries in hybrid systems with a GPU that uses *CUDA* and CPUs.

Dependent Variables

Average learning execution time (s), average inference execution time (s), CPU temperature, GPU temperature, CPU utilization tax (%), GPU utilization tax (%).

Independent Variables

Code compilation, library parallelization capacity, algorithm complexity contained in the libraries, room temperature and execution environment (notebook).

Hypothesis Formulation

The research questions for this experiment are: Does the libraries have similar execution time? Does the execution of algorithms that use libraries to generate a similar temperature in the GPU? Does the execution of algorithms that use libraries to generate a similar temperature in the CPU? Does the framework present the same utilization tax in the GPU? Does the library present the same utilization tax in the CPU? Each of these questions must be answered for both learning and inference process.

All the questions are answered from data extracted from execution of the same artificial neural network. For the first question, it can be considered learning average time of the network (μ^{lrn}) and the inference average time (μ^{inf}), for the second question it can use the GPU average temperature during the learning process (τ^{lrn}) and during the inference process (τ^{inf}), for the third question it can use the CPU average temperature during the learning process (ψ^{lrn}) and during the inference process (ψ^{inf}), for the fourth question it can utilize the GPU utilization tax average during the learning process (ω^{lrn}) and during the inference process (ω^{inf}) and, for the last question, it can use the CPU utilization tax average (in percentage) during the learning process (ϕ^{lrn}) and during the inference process (ϕ^{inf}). Each one of the measures has to be done for each one library: *TensorFlow* and *PyTorch*. In this context, the following hypothesis can be verified (knowing that, TF = *TensorFlow*, PT = *PyTorch*):

Hypothesis 1 (For Inference and Learning Processes)

H0: There is not statistical differences between the library execution average times, $\left(\mu_{TF}^{i} = \mu_{PT}^{i}\right)$

H1: There are statical differences in the average execution time of the libraries, that means, $\left(\mu_{TF}^{i} \neq \mu_{PT}^{i}\right)$

Hypothesis 2 (For the Learning and Inference Processes)

H0: There is not statistical differences between the GPU average temperature during the library executions, that means, $\left(\tau_{TF}^{i} = \tau_{PT}^{i}\right)$

H1: There are statistical differences between the GPU average temperature during the library executions, that means, $\left(\tau_{TF}^{i} \neq \tau_{PT}^{i}\right)$

Hypothesis 3 (For the Learning and Inference Processes)

H0: There is not statistical differences between the CPU average temperature during the library executions, that means, $\left(\psi_{TF}^{i} = \psi_{PT}^{i}\right)$

H1: There are statistical differences between the CPU average temperature during the library executions, that means, $\left(\psi_{TF}^{i} \neq \psi_{PT}^{i}\right)$

Hypothesis 4 (For the Learning and Inference Processes)

H0: There is not statistical differences between the GPU utilization tax average during the library executions, that means, $\left(\omega_{TF}^{i} = \omega_{PT}^{i}\right)$

H1: There are statistical differences between the GPU utilization tax average during the library executions, that means, $\left(\omega_{TF}^{i} \neq \omega_{PT}^{i}\right)$

Hypothesis 5 (For the Learning and Inference Processes)

H0: There is not statistical differences between the CPU utilization tax average during the library executions, that means, $\left(\phi_{TF}^{i} = \phi_{PT}^{i}\right)$

H1: There are statistical differences between the CPU utilization tax average during the library executions, that means, $\left(\phi_{TF}^{i} \neq \phi_{PT}^{i}\right)$

Object Selection

The experiment utilized a set of image data, the Modified National Institute of Standards and Technology database (MNIST database) (LeCun and

Cortes, 2010). MNIST is a set of manuscript digits commonly used to train image processing systems and has the following features:

- Images of size 28×28 pixels
- 10 classes, one class for each digit
- Subset with 60000 training data
- Subset with 10000 test data

About the codes, the experiment utilized two codes in a *LeNet* CNN implemented in Python, the *LeNet* network is presented by Lecun *et al*. (1998). The first code, using *TensorFlow* libraries, was adapted from the code available in a *TensorFlow* repository in *Github* (GBT, 2017). The second code, used libraries from *PyTorch* was adapted from the code available in a *PyTorch* repository in *Github* (*PyTorch*, 2018b). The adaptations were done in order to let both codes with the same parameter patterns, this process is explained in the subsection 5.1.

Experiment Project

The experiment project can be summarized in the following steps:

1. Preparing the execution environment
2. Adapting the available codes from *TensorFlow* and *PyTorch* repositories
3. Implementing scripts
4. Measuring 100 times the algorithm execution time during the learning phase for each one of the libraries
5. Measuring 100 times the algorithm execution time during the inference phase for each one of the libraries
6. Collecting 100 times the CPU and GPU utilization during the execution of each phase (learning and inference)
7. Collecting 100 times the CPU and GPU temperature during the execution of each phase (learning and inference)
8. Applying statistical tests for the hypothesis analysis

Instrumentation

The used software are the Python environment 3.5.2, *Anaconda* 3, *TensorFlow* 1.8, *PyTorch* 1.0, *CUDA* 9.0, *cuDNN* 7.1, *Nvidia Driver* 396.26, *Nvidia System Management Interface*, *Sensors 3.4.0* and *Ubuntu OS 16.04 LTS*. The hardware for experiment execution is a *Dell Inspiron 15 Gaming i15-7567* notebook, 8GB 2133MHz DDR4 RAM Memory, *7th Generation Intel Core i5-7300HQ Quad Core* (6MB Cache, up to 3.5GHz, 1TB 5400 rpm HD with 8GB cache and *GPU NVIDIA GeForce GTX 1050*. The GPU specifications are available in Table 1.

Table 1: GPU NVIDIA GeForce GTX 1050 Specifications

Technical Information	Value
GPU Architecture	Pascal
NVIDIA Cuda Cores	768
Frame Buffer	4GB GDDR5
Memory Velocity	7G bps
Boost Clock	1392 MHzx

Operation of the Experiment

Preparation

The experiment preparation had as first task the installation of software inside the notebook including the environment *Python*, *Anaconda*, *cuDNN* and the libraries. The selection of a dataset was done after that, *MNIST* dataset was selected by because it is the most used in the related works for library evaluation.

It was selected codes in the libraries official repositories that train and test a *LeNet* CNN network using the MNIST dataset. The code for *PyTorch* library available in *PyTorch* (2018b) and the code for *TensorFlow* library were modified and, posteriorly, used for collection of data. The accomplished modifications aimed to standardize the parameters of both codes according to Table 2.

It was created scripts in order to automatize the collection of data, each script was developed to execute the learning and inference and store the data of each one of the variables.

Execution

The created scripts for automation of data collecting worked in the following way: Each script executed learning and inference of the *LeNet* network sequentially, in each one of these executions it was stored 100 data from two variables with the same criteria.

The execution steps were:

1. Script execution that stores 100 measures of execution time of learning and 100 measures of execution time of inference using *TensorFlow*
2. Script execution that stores 100 measures of temperature during the execution of learning and 100 measures of temperature during the execution of inference using *TensorFlow*
3. Script execution that stores 100 measures of CPU utilization during the execution of learning and 100 measures of CPU utilization during the execution of inference using *TensorFlow*
4. Script execution that stores 100 measures of GPU utilization during the execution of learning and 100 measures of CPU utilization during the execution of inference using *TensorFlow*
5. Repeat the process, but using *PyTorch*

Table 2: Used *LeNet* CNN parameters

Parameters	Value
Batch size	100.00
Number of epochs	1.00
Number of steps	600.00
Learning rate	0.01

Observation about the execution:

- When storing the execution time data, the first value was ignored in order to avoid outliers occasioned by first library calls from libraries
- The environment temperature during execution of *TensorFlow* was 22°C
- The environment temperature during execution of *PyTorch* was among 24°C and 26°C.

Data Collection

In order to collect time data it was utilized the function *time.time()* from *time* library of *Python* language. In order to measure the learning execution time, the time starts to be counted before the training procedure call, the count is finalized after the procedure end. For measuring the inference execution time, the library *time.time()* was applied the same way, but for the classification procedure.

The CPU temperature and utilization collection were done using Sensors 3.4.0, the GPU temperature and utilization rate were done using the *NVIDIA System* Management Interface. The measures were done several times during the execution of each procedure (learning and inference), it can have few measures done before and after the execution of the procedures.

For each one of the codes, one using *TensorFLow* and other using *PyTorch*, it was collected the following dependent samples:

- 100 samples of learning execution time
- 100 samples of inference execution time
- 100 samples of GPU temperature (°C) during the learning phase
- 100 samples of GPU temperature (°C) during the inference phase
- 100 samples of CPU temperature (°C) during the learning phase
- 100 samples of CPU temperature (°C) during the inference phase
- 100 samples of GPU utilization rate (%) during the learning phase
- 100 samples of GPU utilization rate (%) during the inference phase
- 100 samples of CPU utilization rate (%) during the learning phase
- 100 samples of CPU utilization rate (%) during the inference phase

Data Validation

As a way to validate data and to evaluate statistical the raised hypothesis, the statistical test Kolmogov-Smirnov (KS) was used initially for testing if the acquired metrics had a Gaussian probability distribution (normal). From the result of this test (that showed the data had not a normal distribution), it was used the paired *Wilcoxon* test for analysis of the presented hypothesis.

The paired *Wilcoxon* test was selected for analysis of the presented hypothesis because it is a non-parametric test used to compare if the position measures of two samples are equals in case that samples are dependents.

The subsequent section presents the result of KS test and also the obtained results from paired *Wilcoxon* test having in mind the raised hypothesis.

Results

As mentioned at the end of the last section, the KS statistical test was used for verifying the normality of data, therefore, a level of thrust of 95% was applied. From the results it was identified that for all dependent variables, the probability distribution is not normal, because the returned p-values (a measure that indicates the probability of evaluated set to follow the normal distribution), were next to zero (less than 10^{-10}).

As also previously mentioned, the used test for hypothesis analysis was the paired *Wilcoxon* test with a thrust level of 95%, we will analyze separately its results in subsection 6.1.

Analysis and Interpretation

Execution Time (Hypothesis 1)

Here we analyze data for Hypothesis 1.

The result of paired *Wilcoxon* test for the learning execution time returned a p-value of $3.896559845095909\times10^{-18}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 4, that shows time in seconds for each code, it shows clearly a smaller learning execution time for the code that uses *PyTorch* library.

The result of paired *Wilcoxon* test for the inference execution time returned a p-value of $3.896559845095909\times10^{-18}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 5, that shows time in seconds for each code, shows clearly a smaller inference execution time using the *PyTorch* library.

GPU Temperature (Hypothesis 2)

Here we will analyze data for Hypothesis 2.

The result of paired *Wilcoxon* test for GPU average temperature during learning phase returned a p-value of

$3.896559845095909\times10^{-18}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 6, shows temperature in Celsius degrees (°C) for each code, shows clearly a smaller average temperature for the code that uses *PyTorch* library.

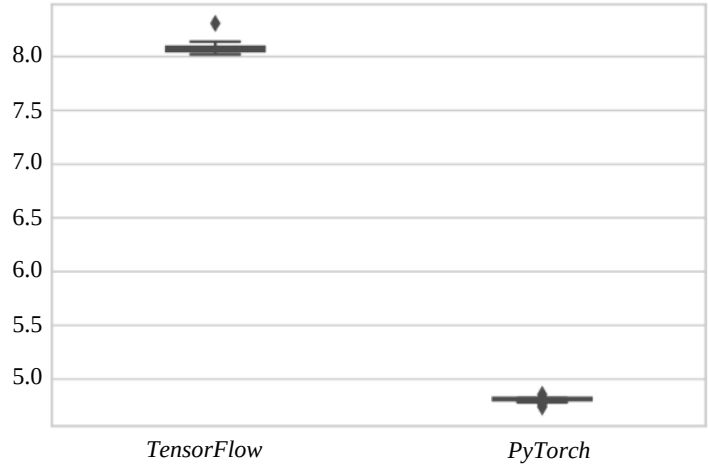

Fig. 4: Learning execution time

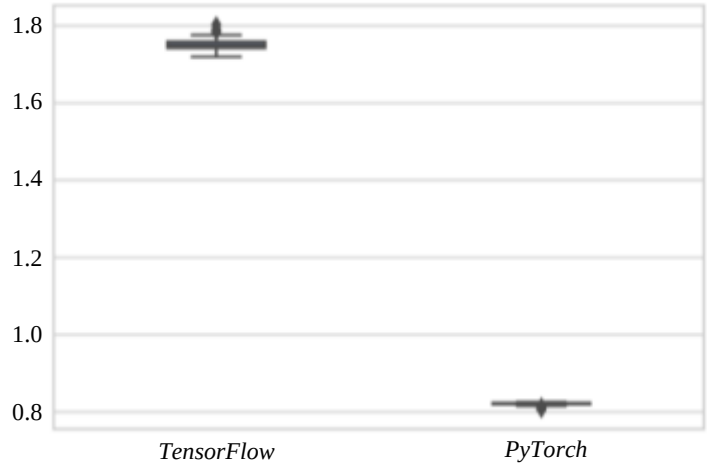

Fig. 5: Inference execution time

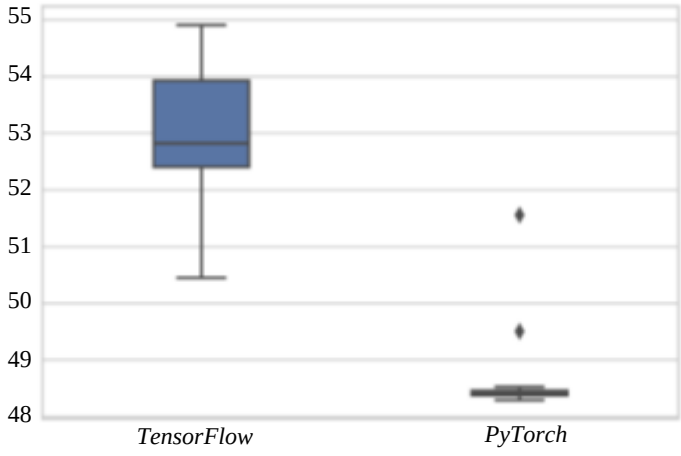

Fig. 6: GPU temperature (°C) during learning phase

Another important thing that can also be noted analyzing the graphic in Fig. 6 is the presence of two outliers for *PyTorch*. They may be occasioned for a sudden drop during the begin of the code execution that uses *PyTorch* for learning, that means, the GPU was more heated before the execution of *PyTorch*.

The result of paired *Wilcoxon* test for GPU average temperature during the inference phase returned a p-value of $9.595359087770175\times10^{-16}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 7, shows temperature in Celsius degrees (°C) for each code, shows clearly a smaller average temperature for the code that uses *PyTorch* library.

Another important thing that can also be noted analyzing the graphic in Fig. 7 is the presence of some outliers for *PyTorch*. They may be occasioned for a sudden drop during the begin of the code execution that uses *PyTorch* for learning, that means, the GPU was more heated before the execution of *PyTorch*.

CPU Temperature (Hypothesis 3)

The result of paired *Wilcoxon* test for CPU average temperature during the execution of the learning phase returned a p-value of $3.6318189005086153\times10^{-13}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 8, shows temperature in Celsius degrees (°C) for each code, shows clearly a smaller average temperature for the code that uses *PyTorch* library.

Another important thing that can also be noted analyzing the graphic in Fig. 8 is the presence of some outliers for *TensorFlow* and *PyTorch*. The outliers from *PyTorch* could also be occasioned for a sudden drop of temperature during the begin of learning code execution, it means, the GPU was more heated before the execution of *PyTorch*. The outliers from *TensorFlow*, contrary, are occasioned by a sudden ascent of the CPU temperature when executing *TensorFlow*.

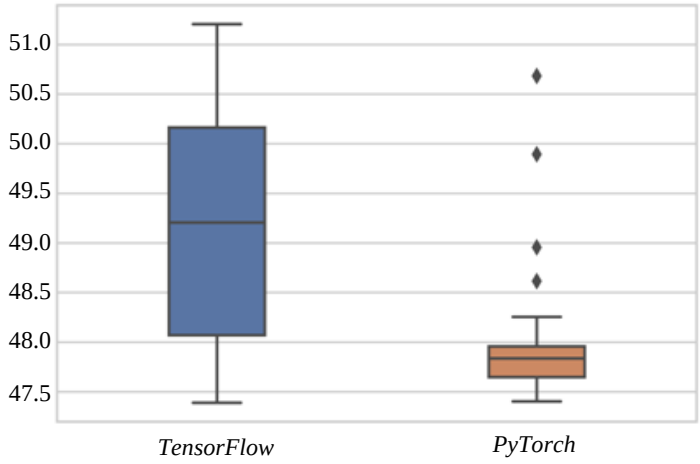

Fig. 7: GPU Temperature (°C) during inference phase

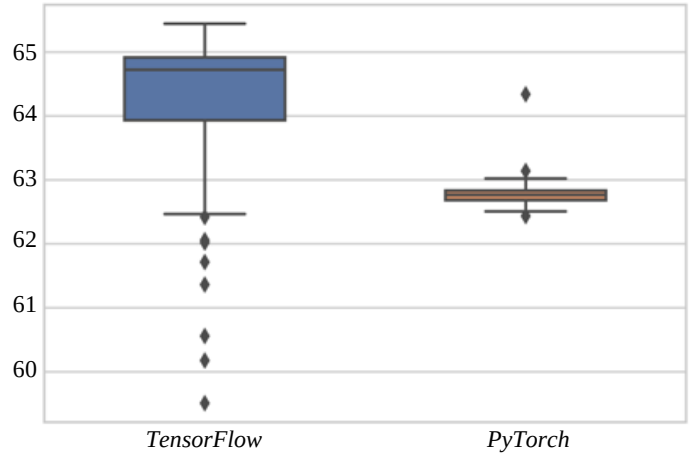

Fig. 8: CPU Temperature (°C) during learning phase

The result of paired *Wilcoxon* test for CPU average temperature during the execution of the inference phase returned a p-value of $4.531861061109546\times10^{-18}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 9, shows temperature in Celsius degrees (°C) for each code, shows clearly a smaller average temperature for the code that uses *PyTorch* library.

Another important thing that can also be noted analyzing the graphic in Fig. 9 is the presence of some outliers. The graphic presents some outliers of low temperature, both for *TensorFlow* and for *PyTorch*, that could be occasioned by a sudden ascent of temperature. the generated outliers in the *PyTorch* data could also be occasioned by oscillation in CPU utilization.

GPU Utilization (Hypothesis 4)

The result of paired *Wilcoxon* test for GPU average utilization rate during the execution of the learning phase returned a p-value of $3.896559845095909\times10^{-18}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 10, that presents the GPU utilization rate (%) for each code, shows clearly a bigger utilization rate during the learning phase execution from code that uses *TensorFlow* library.

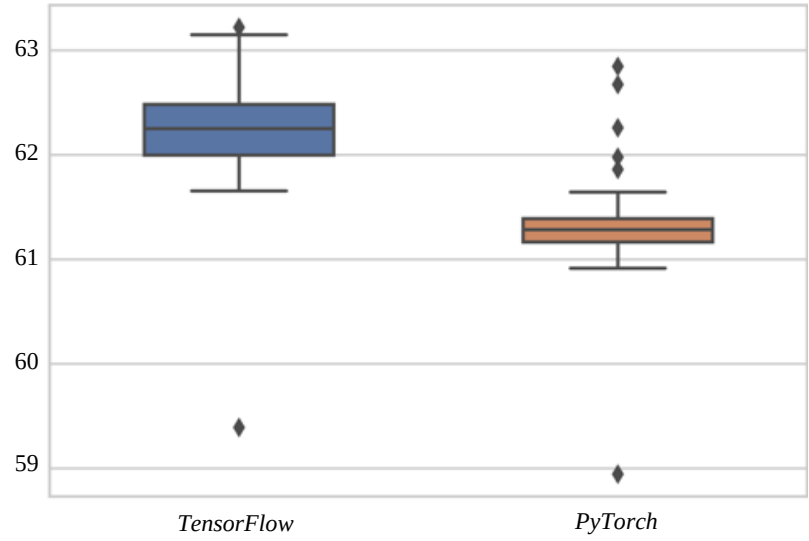

Fig. 9: CPU Temperature (°C) during inference phase

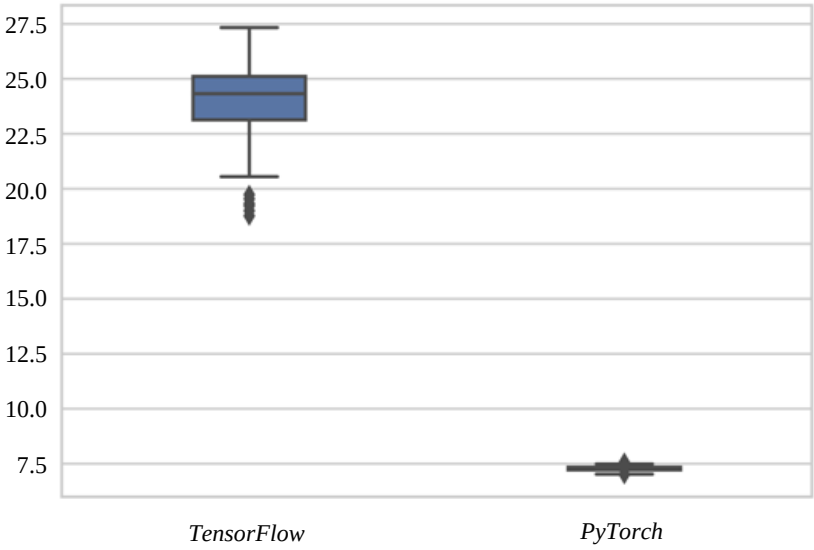

Fig. 10: GPU Utilization rate(%) during learning phase

The result of paired *Wilcoxon* test for GPU average utilization rate during the execution of the inference phase returned a p-value of $3.896559845095909\times10^{-18}$, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 11, that presents the GPU utilization rate (%) for each code, shows clearly a bigger utilization rate during the learning phase execution from code that uses *TensorFlow* library.

CPU Utilization (Hypothesis 5)

The result of paired *Wilcoxon* test for the CPU utilization rate during execution of learning phase returned a p-value of 0.4851894147442428, therefore the hypothesis H0 was not rejected. The graphic in Fig. 12, that presents a CPU utilization rate (%) for each code, shows a utilization rate very similar, with the *PyTorch* rate presenting a bigger variance.

The result of paired *Wilcoxon* test for the CPU utilization rate during execution of inference phase returned a p-value of 3.3133542319431908^{-18}, therefore the hypothesis H0 was strongly rejected and, consequently, the hypothesis H1 was not rejected. The graphic in Fig. 13, that presents the CPU utilization rate (%) for each code, shows clearly a bigger utilization rate during inference phase of code using *PyTorch* library.

Threats to Validity

Statistical tests were used as a way to mitigate bias related to conclusion with respect to established hypothesis (conclusion validity). The first was the KS test, using for verification of data normality. This step was necessary for choosing the following test, related to comparison of algorithms to be utilized, that, being a rejected null hypothesis of the KS test, it was used a non-parametric test for dependent samples, the paired *Wilcoxon* test. Thus, it might have a statistically satisfactory conclusion, avoiding a selection of a library detriment of another only by the sample averages.

It was monitoring the environment temperature from the Aracaju city during the data collection process as a way of mitigating bias about the temperature data samples (intern validity), but the intern temperature of the room where the notebook was located was not adequately monitored, thus being a threat to validity.

Another threat to validity about the intern validity is that it was not monitored all the process executed by the operating system during the code execution. Some processes might occasionally influence in the code execution causing outliers. It was mitigated for execution times, eliminating the collection of the first sample and was mitigated for all variables turning the graphic environment off and using the Anaconda software.

Only one kind of deep network was used, the *LeNet* CNN, what represent a low variety of networks for library analysis constituting a threat to external validity. Another threat to external validity is that only one device was used to processing the codes, a greater variety of devices could mitigate the bias.

The experiment authors have little experience using both the libraries, therefore the codes may have errors that were unnoticed constituting a threat to **construct validity**. In order to mitigate this threat it was used codes already done available in the official repositories from the library developers (with modifications, as previously explained), it was used a widely used network (*LeNet* CNN) in literature and am also widely used among the related work dataset (MNIST dataset).

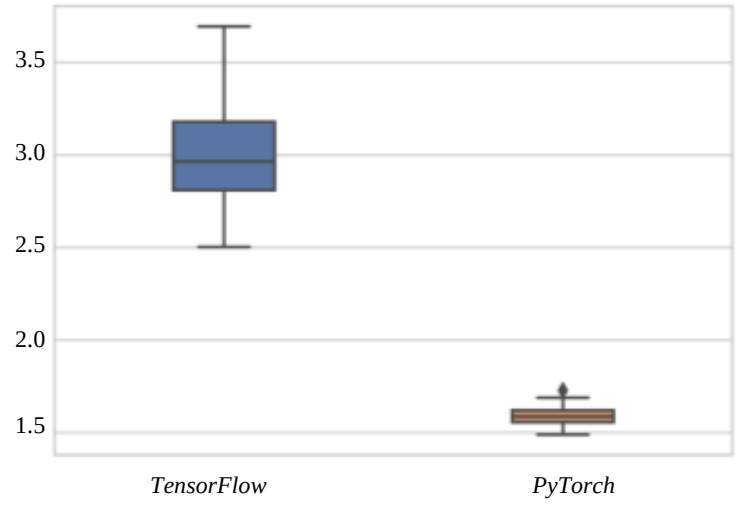

Fig. 11: GPU utilization rate (%) during inference phase

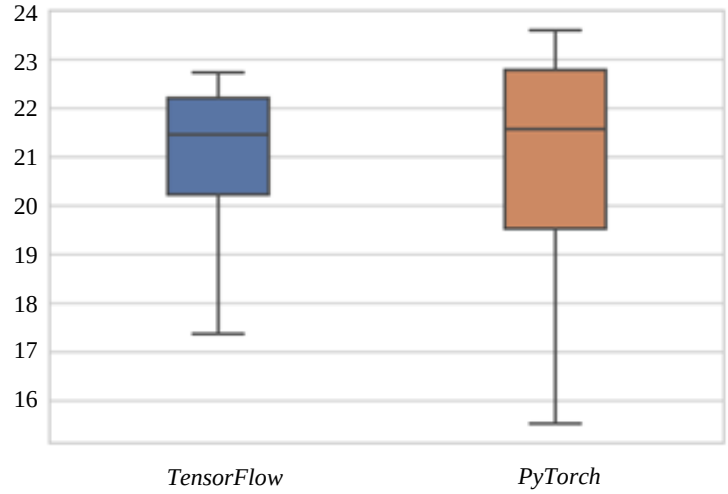

Fig. 12: CPU utilization rate (%) during learning phase

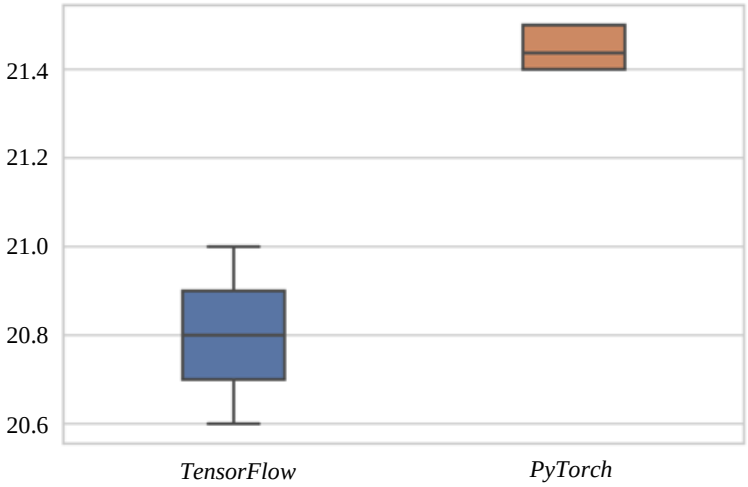

Fig. 13: CPU utilization rate (%) during inference phase

Conclusion and Future Works

Deep Neural Network tools are a recurrent subject in the academia and industry with several researches related to the creation of new libraries, library performance evaluation, execution optimization techniques and the creation of dedicated devices.

In this context, the present work showed an approach related to performance of available open source libraries, comparing them in a heterogeneous architecture with CPU and GPU according to their performance.

The compared libraries *TensorFlow* and *PyTorch* are indicated tools when dealing with CNN implementation with CUDA support. In order to compare them, execution time, hardware temperature impact and utilization tax were evaluated.

In order to test the hypothesis (about execution time, hardware temperature and hardware utilization), it was used execution time in seconds, CPU and GPU temperature in degrees Celsius and CPU and GPU utilization tax in percentage. Two codes were used, the first one using *PyTorch* library and the second one using *TensorFlow* framework, both of them implemented in a *LeNet* CNN and being executed with the MNIST dataset.

It was possible verify that *PyTorch* library presented a better performance in the showed context. However, it is necessary to note that *PyTorch* uses less the GPU potential than it does with CPU potential, unlike *TensorFlow* that during inference focus its processes in the GPU.

Some theses may be raised for a better performance of *PyTorch*. The first one is that it uses less GPU, reducing the communication bottleneck between CPU and GPU. The second one is that *PyTorch* utilizes function less automatized leaving harder work for programmers. *TensorFlow* is much simpler and intuitive, however it may have impacted in algorithm generalization and, consequently, in non-optimized algorithms for specific neural networks.

As future work, it is recommended a deeper investigation into the impact of each library in GPUs utilizing other hardware as clusters with several GPUs. It also is recommended verifying the performance of these libraries implementing other CNN and RNN networks. Lastly, it is recommended comparing these libraries with other popular libraries such as *Caffe*, *Microsoft CNTK*, *Theano* and *Deeplearning4j*.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

The authors thank the Programa de Pós-Graduação em Ciência da Computação da UFS (PROCC-UFS) for their support..

Author's Contributions

All authors are equally contributed in this work and this paper.

Mr. Felipe Florencio: Participated in all experiments, coordinated the data analysis and contributed to the writing of the manuscript.

Mr. Thiago Valença: Participated in all experiments, coordinated the data analysis and contributed to the writing of the manuscript.

Dr. Edward David Moreno: Coordinated the experiments and contributed to the writing of the manuscript.

Dr. Methanias Colaço Junior: Designed the research plan, organized the study and contributed.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo and Z. Chen *et al*., 2015. *TensorFlow*: Large-scale machine learning on heterogeneous systems. *tensorflow*.org

Bahrampour, S., N. Ramakrishnan, L. Schott and M. Shah, 2015. Comparative study of *Caffe*, neon, *theano* and *torch* for deep learning. CoRR.

Basili, V.R., G. Caldiera and H.D. Rombach, 1994. The goal question metric approach. Encyclopedia of Software Engineering, Wiley.

da Costa e Silva Franco, A., 2016. On deeply learning features for automatic person image reidentification. PhD Thesis, Mechatronics Federal University of Bahia, Salvador, Brazil.

Deng, L., 2014. A tutorial survey of architectures, algorithms and applications for deep learning. APSIPA Trans. Signal Inform. Process. DOI: 10.1017/atsip.2013.9

Dertat, A., 2017. Applied deep learning – part 4: Convolutional neural netwotk. https://bit.ly/2QCu72R

Fonnegra, R.D., B. Blair and G.M. Diaz, 2017. Performance comparison of deep learning frameworks in image classification problems using convolutional and recurrent networks. Proceedings of the IEEE Colombian Conference on Communications and Computing, Aug. 16-18, IEEE Xplore Press, Cartagena, Colombia, pp: 1-6. DOI: 10.1109/ColComCon.2017.8088219

Fukushima, K., 1979. Self-organization of a neural network which gives position-invariant response. Proceedings of the 6th International Joint Conference on Artificial Intelligence, (CAI' 79), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp: 291-293.

Goldsborough, P., 2016. A tour of *tensorflow*. CoRR.

Goodfellow, I., Y. Bengio and A. Courville, 2016. Deep learning. MIT Press.

GBT, 2017. Convolutional neural network estimator for MNIST, built with tf.layers. Google Brain Team.

GBT, 2018a. A guide to TF layers: Building a convolutional neural network. Google Brain Team. https://www.*tensorflow*.org/tutorials/layers

GBT, 2018b. *Tensorflow*: An open source machine learning library for research and production. Google Brain Team. https://www.*tensorflow*.org

Hamed Habibi Aghdam, E.J.H.A., 2017. Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification. 1st Edn., Springer International Publishing, pp: 282.

Kovalev, V., A. Kalinovsky and S. Kovalev, 2016. Deep learning with *theano*, *torch*, *caffe*, *tensorflow* and deeplearning4j: Which one is the best in speed and accuracy? Proceedings of the 13th International Conference on Pattern Recognition and Information Processing, Oct. 3-5, At Minsk, Belarus, pp: 99-103.

Kruchinin, D., E. Dolotov, K. Kornyakov, V. Kustikova and P. Druzhkov, 2015. Comparison of Deep Learning Libraries on the Problem of Handwritten Digit Classification. In: Analysis of Images, Social Networks and Texts, Khachay, M.Y., N. Konstantinova, A. Panchenko, D. Ignatov and V.G. Labunets (Eds.), Springer International Publishing, Cham, pp: 399-411.

Lecun, Y., L. Bottou, Y. Bengio and P. Haffner, 1998. Gradient-based learning applied to document recognition. Proc. IEEE, 86: 2278-2324. DOI: 10.1109/5.726791

LeCun, Y. and C. Cortes, 2010. MNIST handwritten digit database.

Liu, L., Y. Wu, W. Wei, W. Cao and S. Sahin *et al*., 2018. Benchmarking deep learning frameworks: Design considerations, metrics and beyond. Proceedings of the IEEE 38th International Conference on Distributed Computing Systems, Jul. 2-6, IEEE Xplore Press, Vienna, Austria, pp: 1258-1269. DOI: 10.1109/ICDCS.2018.00125

Parvat, A., J. Chavan, S. Kadam, S. Dev and V. Pathak, 2017. A survey of deep-learning frameworks. Proceedings of the International Conference on Inventive Systems and Control, Jan. 19-20, IEEE Xplore Press, Coimbatore, India, pp: 1-7. DOI: 10.1109/ICISC.2017.8068684

Paszke, A., S. Gross, S. Chintala, G. Chanan and E. Yang *et al*., 2017. Automatic differentiation in *pytorch*.

PyTorch, 2018a. *Pytorch*: About. https://*pytorch*.org/about/

PyTorch, 2018b. *Pytorch*/examples. https://github.com/*pytorch*/examples/tree/master/mnist

Raschka, S., 2015. Python Machine Learning. 1st Edn., Packt Publishing, Birmingham, ISBN-10: 1783555149, pp: 454.

Shatnawi, A., G. Al-Bdour, R. Al-Qurran and M. Al-Ayyoub, 2018. A comparative study of open source deep learning frameworks. Proceedings of the 9th International Conference on Information and Communication Systems, Apr. 3-5, IEEE Xplore Press, Irbid, Jordan, pp: 72-77. DOI: 10.1109/IACS.2018.8355444

Shi, S., Q. Wang, P. Xu and X. Chu, 2016. Benchmarking state-of-the-art deep learning software tools. Proceedings of the 7th International Conference on Cloud Computing and Big Data, Nov. 16-18, IEEE Xplore Press, Macau, China pp: 99-104. DOI: 10.1109/CCBD.2016.029

Verhelst, M. and B. Moons, 2017. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to IOT and edge devices. IEEE Solid-State Circuits Magazine, 9: 55-65. DOI: 10.1109/MSSC.2017.2745818